DOCUMENT RESUME

ED 367 296 IR 016 556

AUTHOR Moursund, Dave; Yoder, Sharon
TITLE Problem Solving and Communication in a "HyperCard" Environment.
INSTITUTION International Society for Technology in Education, Eugene, OR.
REPORT NO ISBN-1-56484-036-0
PUB DATE 93
NOTE 167p.
AVAILABLE FROM International Society for Technology in Education, 1787 Agate St., Eugene, OR 97403-1923.
PUB TYPE Guides - General (050)

EDRS PRICE MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS *Communication (Thought Transfer); *Computer Assisted Instruction; Computer Software; Educational Environment; Experiential Learning; Feedback; Higher Education; *Hypermedia; Journal Writing; Learning Activities; Multimedia Instruction; *Problem Solving; Programming; Teacher Education
IDENTIFIERS Apple MacIntosh; *HyperCard; Learning Environments

ABSTRACT

This book provides an introduction to HyperCard, with the main focus being on communication and problem solving. The book covers most of the main features of HyperCard and includes a very brief introduction to HyperTalk. This book is specifically for people who want to move toward higher levels of HyperCard knowledge and skills. It is designed for use in a short course or workshop on communication and problem solving in a HyperCard environment; additional uses are as a supplementary text that emphasizes communication and problem solving within an extensive course on HyperCard and as a text in inservice and preservice teacher education settings. The book contains activities to help guide individuals through the process of journal writing, using the computer, and obtaining feedback. The book is composed of 11 chapters with appendices on keyboard shortcuts, visual effects, and research on problem solving. The chapters are as follows: (1) Introduction and Overview; (2) What is Problem Solving? (3) Getting Started in HyperCard; (4) Making Cards and Stacks; (5) Paint Tools; (6) Buttons; (7) Fields; (8) Backgrounds; (9) Stack Design Issues; (10) Scripting Beeps and Music; and (11) What Else is There in HyperCard? (Contains 25 references.) (TMK)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED 367 296

Problem Solving and Communication in a HyperCard Environment

12016 556

BEST COPY AVAILABLE

Dave Moursund and Sharon Yoder

"PERMISSION TO REPRODUCE THIS MATERIAL IN **OTHER THAN PAPER COPY** HAS BEEN GRANTED BY

D. Moursund

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)"

About the Authors

Dr. David Moursund has been teaching and writing in the field of computers in education since 1963. He is a professor in the College of Education at the University of Oregon. Dr. Moursund has authored or co-authored more than 25 books and numerous articles on computers and education. He was the chairman of the department of computer science at the University of Oregon from 1969 to 1975.

In 1979 he founded the International Council for Computers in Education (ICCE), which became the International Society for Technology in Education (ISTE) in 1989 when it merged with the International Association for Computing in Education. Dr. Moursund is currently the executive officer of ISTE.

Dr. Sharon Yoder has taught mathematics and computer science at the junior high and high school level for 15 years. Her most recent public school experience was as a secondary computer science teacher and computer coordinator involved in developing system-wide computer curriculum and in planning teacher inservice training.

In addition, she has taught mathematics, computer science, and computer education at a number of universities in northeastern Ohio, including Kent State University, the University of Akron, and Cleveland State University. She currently teaches computer education courses at the University of Oregon.

For the past several years, Dr. Yoder has conducted workshops and presented papers at national conferences. She has been involved in a number of book-publishing projects, including the *Nudges* series, *An Introduction to Programming in Logo Using LogoWriter*, and *An Introduction to Programming in Logo Using Logo PLUS*. In addition, she has been a frequent contributor to *The Computing Teacher*. She is currently editor of *Logo Exchange*.

Editing: Christy McMannis and Ron Renchler, The Electronic Page
Editorial Assistance: Anita Best
Production Assistance: Jean Hall
Book Cover: Tamara Kidd

© International Society for Technology in Education, 1993
1787 Agate Street, Eugene, Oregon 97403-1923
Order Desk: 800/336-5191
Phone: 503/346-4414
Fax: 503/346-5890
CompuServe: 70014,2117
Internet: ISTE@Oregon.uoregon.edu

ISBN 1-56484-036-0

Problem Solving and Communication in a *HyperCard* Environment

Dave Moursund and Sharon Yoder

© International Society for Technology in Education, 1993

Table of Contents

Preface i

Possible Uses of This Book ii
What Do You Need to Know to Use This Book? ii
Hardware and Software Needed to Use This Book iii

Chapter 1. Introduction and Overview 1

Computers and Computer Programming 1
What Is *HyperCard*? 1
HyperTalk Is a Programming Language 4
HyperCard Is a Versatile Tool 5
Reading and Writing Hypermedia 5
Objects in *HyperCard* 7
Aids to Problem Solving 7
The Human Mind 8
Educational Uses of *HyperCard* 8
How to Use This Book 9
Activities 10

Chapter 2. What Is Problem Solving? 13

Problem Situations 13
Definition of a Formal Problem 15
A Five-Step Plan for Attacking a Problem 17
Domain Specificity 18
Transfer of Learning 18
Activities 19

Chapter 3. Getting Started in *HyperCard* 21

A First Look at *HyperCard* 21
Getting Familiar With *HyperCard* 22
Problem Solving—A Thought Experiment 24
Why Learn to Create Stacks? 25
Activities 26

Chapter 4. Making Cards and Stacks 29

Setting the User Level 29
Creating a New Stack 30
Cards 31
Creating Additional Cards 32
Using the Tool Window 33
The Paint Tools 35

Copying and Pasting a Card 36
Deleting a Card and Cutting a Card 36
Navigation in a Stack 36
Saving a Stack 38
The Idea of a Primitive 38
Learning *HyperCard* 39
Bugs and Debugging 40
Learning in a *HyperCard* Environment 40

Chapter 5. Paint Tools 43

The Tool Window 43
Graphical User Interface (GUI) 45
Using the Paint Tools 45
The Select Tool 45
The Lasso Tool 46
The Pencil Tool 46
The Brush Tool 48
The Eraser Tool 48
The Line Tool 49
The Spray Can Tool 49
The Rectangle Tool 50
The Round Rectangle Tool 51
The Bucket Tool 51
The Oval Tool 52
The Curve Tool 52
The Text Tool 53
The Regular Polygon Tool 54
The Polygon Tool 54
Clip Art 56
Menus 57
Problem Solving 60
Activities 60

Chapter 6. Buttons 63

Creating a Button 63
Button Icons 67
Copying, Pasting, and Deleting a Button 67
Button Conventions 68
Scripting Buttons 69
Visual Effects 72
Consistency 73
A Button Is an Object 74
Nonsequential Communication 74
Detecting and Correcting Bugs 75
Problem Solving 76
Activities 77

Chapter 7. Fields 79
Creating a Field 79
Entering Text Into a Field 80
Field Properties 80
Cutting and Pasting From Other Documents 81
Objects Are Layered 81
Pop-Up Field 83
Problem Solving 84
Activities 85

Chapter 8. Backgrounds 89
Using Backgrounds 89
Creating a Background 89
New Background 93
Buttons and Fields on a Background 94
Layering Buttons and Fields 94
Finding Lost Buttons and Fields 95
Communication 96
Activities 96

Chapter 9. Stack Design Issues 99
Stack Structure 99
Linear and Jump Linear Stacks 100
Tree Stacks 101
Single-Frame Stacks 104
Using Visual Effects 105
Scroll 105
Dissolve 105
Wipe 106
Iris/Zoom 106
Barn Door 106
Checkerboard/Venetian Blinds 106
HyperCard 2.0 Effects 106
Backgrounds 106
Conventions for Using Icons 108
Some Other Considerations 111
Tiny Cards—An Aid to Creating a Stack Map 112
Activities 113

Chapter 10. Scripting Beeps and Music 115
The Beep Command 115
The Message Box 116
The Play Command 118
Representing Notes 119
Using the Play Command 121

A Button With Music 122
Making Mistakes While Using the Play Command 123
Bugs and Debugging 123
Sound as Part of the Stack Design 124
Activities 124

Chapter 11. What Else Is There in *HyperCard*? 127

Programming Languages 127
Modeling (Representing) Problems 128
Mental Models 128
Written Models 128
Scale Models 129
Mathematical Modeling 129
Computer Models 129
Animation by Flipping Cards 130
Interactive Programming 131
Programming Language Constructs 133
Activities 134

Appendix A. Keyboard Shortcuts 135

Appendix B. Visual Effects 139

Appendix C. Research on Problem Solving 141

References 151

Index 153

Preface

This book provides an introduction to *HyperCard*, with the main focus being on *communication* and *problem solving*. The book covers most of the main features of *HyperCard* and includes a very brief introduction to HyperTalk, the underlying programming language in the *HyperCard* system.

The *HyperCard* system is designed so that a beginner can easily get started and experience considerable success. However, the *HyperCard* system is a very powerful tool with a huge range of capabilities. Most people find that it takes a great deal of time, effort, and experience to master this tool.

Communication: To create a *HyperCard* document—called a stack—you communicate with a computer system that is running the *HyperCard* software. You create this stack so that you yourself can use it (so that it communicates effectively with you) and/or so that others can use it (so that it communicates effectively with others).

Problem Solving: The computer is a general-purpose aid to solving problems, much like books, calculators, reading, writing, and arithmetic are general aids to problem solving. As you learn *HyperCard*, you can simultaneously learn about problem solving in a *HyperCard* environment, problem solving in a computer environment, and problem solving in noncomputer environments. Thus, you can improve your overall problem-solving skills.

One of the major difficulties in problem solving is that most people have had little formal instruction in the precise representation of problems and in the careful thinking needed to express the details of how to solve a particular type of problem. A second difficulty is that most people are not accustomed to using precise communication. When you tell a computer to do something, the smallest error can lead to incorrect results. For example, suppose that you intend to tell a computer to wait 15 seconds before displaying the next picture in a sequence. However, perhaps you actually key in the number 51 instead of 15. The computer has no way of knowing that you meant 15; it will wait 51 seconds between pictures.

For most people, the initial stage of learning *HyperCard* is one of learning menu choices and mouse operations to accomplish certain specific tasks. As you move from trying to learn the rudiments of *HyperCard* toward mastery of this tool, you will find that your focus will shift from learning keypress sequences and simple statements in the HyperTalk language to more fundamental issues, such as communication (with yourself and with others) and problem solving. This book is

specifically designed for people who want to move toward such higher levels of *HyperCard* knowledge and skills.

Possible Uses of This Book

Problem Solving and Communication in a HyperCard Environment is designed specifically for use in a short course or workshop on communication and problem solving in a *HyperCard* environment.

Because the electronic digital computer is a general-purpose aid to communication and problem solving, any course about computers or computer applications could have a major focus on these two areas. However, very few texts or computer courses emphasize these very important topics. Instead, most texts and courses on *HyperCard* focus specifically on teaching the user about the software program, and they place very little emphasis on either communication or problem solving in a computer environment or in other more general environments.

Therefore, a second possible use of this book is as a supplementary text that emphasizes communication and problem solving within an extensive course on *HyperCard*. When this book is used in that way, the assumption is that you will also use a more conventional *HyperCard* text.

A third possible use of this book is as a text in inservice and preservice teacher education settings. *HyperCard* is a valuable productivity tool for educators and is now widely used by students at all educational levels. Thus, many educators are finding it desirable to learn to use *HyperCard* and to learn to work with students in a *HyperCard* environment. In this case, it is very important that educators understand the underlying concepts related to learning in a *HyperCard* environment and that they know how to help their students learn these concepts. This book provides a way to begin mastering the concepts involved in teaching *HyperCard*.

Most people learn best when they are actively engaged in a hands-on, learn-by-doing environment. You, the reader and learner, need to assume responsibility for creating this environment for yourself. As you use this book, you will want to do extensive writing in a personal journal, you will want to try out many different ideas on a computer, and you will want to obtain feedback on your *HyperCard* products both from yourself and from others. This book includes a number of activities that will help guide you through this process.

What Do You Need to Know to Use This Book?

This book is designed for people who are just beginning their study of *HyperCard*. Remember, the main focus of this book is on communication and problem solving in a *HyperCard* environment. The goal is to help you move beyond learning the rudiments of the *HyperCard* system.

To use this book, you should be reasonably comfortable with using the Macintosh computer system. If you have used a word processor and have created and printed documents, you have the level of experience needed to get started in this book.

Hardware and Software Needed to Use This Book

To run the *HyperCard* program, it is best to have a Macintosh with at least 1 megabyte of memory. While it is possible to use *HyperCard* with two 800K floppy-disk drives, you can quickly find yourself doing a great deal of disk swapping. Thus, it is best if you have a hard drive for your work with *HyperCard*.

Versions of *HyperCard* fall into two categories—those before Version 2.0, and Version 2.0 and after. (In this book we refer to Version 2.0 and after as Version 2.0.) You do not need to have *HyperCard* Version 2.0 to use this book. Nearly all the activities will work exactly as described if you use Version 1.2.5. Features unique to Version 2.0 will always be noted. However, if you have *HyperCard* Version 1.0, 1.1, or 1.2, you will occasionally encounter features mentioned in this book that have been added to Version 1.2.5, and thus they are not be available in these earlier versions. Therefore, it would be best if you acquired at least Version 1.2.5 to work with this book.

The pictures of screens in this book were created using Version 2.0. If you are using an earlier version, your screen will occasionally look somewhat different. In general, the different appearance of the screen will not affect the way the examples work.

Dave Moursund and Sharon Yoder
May 1993

Chapter 1. Introduction and Overview

This chapter contains a brief introduction to the key ideas covered in this book. First we give a little background on *HyperCard* and its underlying programming language, HyperTalk. Then we provide a brief introduction to communication in a computer-based multimedia environment. Finally, we discuss some of the key ideas in problem solving.

Computers and Computer Programming

A computer is a machine designed to carry out a detailed, step-by-step set of instructions rapidly and automatically. This step-by-step set of instructions is called a *computer program*. People write computer programs to help themselves and others solve certain types of problems.

There are many different computer programming languages. The programming language in the *HyperCard* system is called HyperTalk. It is a powerful, versatile, and sophisticated programming language.

In the early history of computers, it took a great deal of talent, training, and practice to learn to write computer programs. However, considerable progress has occurred since then. Now there are many computer programming environments such as *HyperCard* where, by appropriate use of menu choices, it is possible to write quite complex programs. Even grade school children can learn to do this.

In a *HyperCard* environment, two types of computer programming occur. First, there is computer programming that is invisible to the programmer—in fact, it is generally not called programming. This is the type of programming that is done by making menu choices. Most of this book focuses on this type of programming.

A second type of programming occurs when one examines a HyperTalk program—called a script—and adds to or modifies it. When working with scripts, you are able to look at the detailed instructions written in HyperTalk. You can write and edit such instructions. That is, you are doing computer programming in the traditional sense. This book provides only a brief introduction to that type of programming.

What Is *HyperCard?*

HyperCard is a software application that runs on Macintosh computers. Since it was first introduced in 1987, *HyperCard* has become increasingly popular. A variety of *HyperCard*-like products have been introduced for Apple II, Macintosh, and MS-DOS/PC-compatible computers. These

HyperCard-like products have become common productivity tools for a wide range of computer users in business, industry, and government. In addition, there has been a strong movement toward their use at all levels of education by both teachers and students.

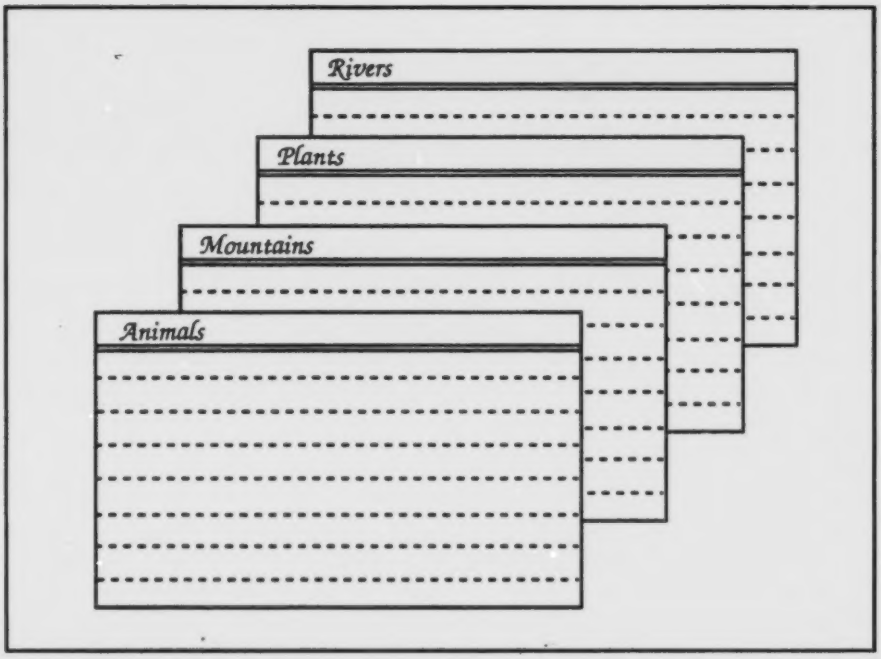
While *HyperCard* and *HyperCard*-like products are relatively new to computing, the underlying concepts behind these products are not new. All of these products allow you to produce nonsequential documents that contain text, graphics, and sound. The idea of nonsequential text has been talked about for many years. The term used to describe such nonsequential text, *hypertext*, was coined by Ted Nelson in his book *Computer Lib/Dream Machines*, published in 1974.

A familiar example of a noncomputer hypertext document is a "choose your own adventure" book. A dictionary and an encyclopedia are additional examples because these reference books are not generally read from beginning to end but rather are accessed nonsequentially.

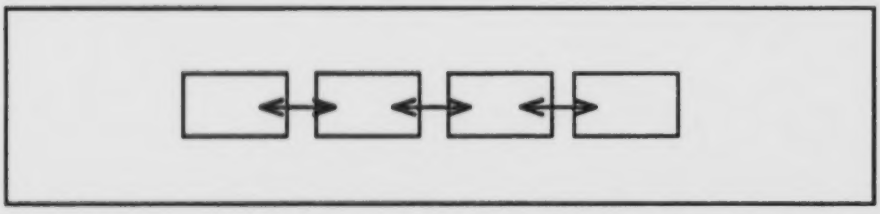
With *HyperCard*, you can use a Macintosh computer to produce your own hypertext documents. *HyperCard* lets you easily add graphics and sound to your documents; you can also access videodiscs and other external media. The combination of text and other media into nonsequential documents produces *hypermedia* documents. *HyperCard* makes creating such documents quite easy.

HyperCard can be thought of as a computer application designed to assist the user in "reading" and "writing" hypermedia documents. In *HyperCard*, the documents are called *stacks*. Whenever you open a *HyperCard* document, you have opened a stack.

What is in a stack? The first and most obvious component of a stack is a *card*. You can think of a *HyperCard* stack as being much like a pile of index cards. Every *HyperCard* stack contains one or more of these cards. Cards are the building blocks out of which you create stacks. Information in *HyperCard* is stored in the cards that make up a stack, just as information put on index cards is stored in a card file or recipe box.



A stack of cards can be put together in a linear fashion, much like the pages of a book. In that format the communication with the user is designed to be "read" in a linear fashion, much like reading a book. In the following stack map (visual representation of a stack), think of starting at the left-most card and moving toward the right. Perhaps, from time to time, you move back to reread a card.

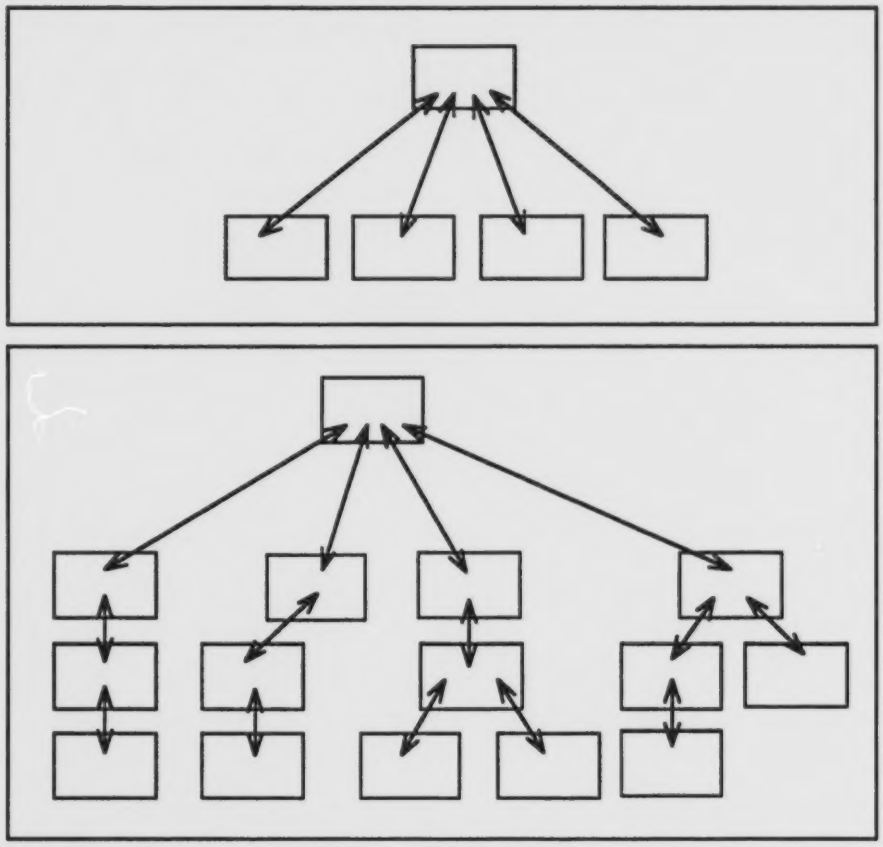


Even in this simple design, a *HyperCard* stack may be much different than the pages of a book. The cards may include sound and visual effects that occur in moving from one card to the next. The cards may contain graphics—even animation—that is activated by clicking on a button on a card.

It is much more complex to design effective communication that takes advantage of putting cards together in a nonlinear fashion. The following two stack maps are examples of nonlinear designs. In the first stack map, the reader starts at the top and makes a decision about which of the other four cards to read. After reading one of them, the reader returns to the top card in the diagram. For example, the first card might contain names of people, while the other cards contain detailed information about the

people. After reading about one person, the reader returns to the first card and has the opportunity to read about a different person.

In the second stack map, the reader may progress further along one of the first four main branches before returning to the top card. This design might be good for containing information about the members of a family tree.



HyperTalk Is a Programming Language

HyperCard is a *computer application*. A computer application is simply a complex computer program that you can use to accomplish some task. If you have experience with the Macintosh, you have used other computer applications and you will have little difficulty getting started with *HyperCard*. You can produce quite sophisticated hypermedia documents by using the pull-down menus that appear at the top of the screen when you start *HyperCard*. From this point of view, *HyperCard* is like many other computer applications, such as a word processor, database, or graphics application. You already know how to write, organize data, and draw. Computer applications augment your capabilities in these areas.

However, in addition to its many menu-based features, *HyperCard* has a built-in programming language called HyperTalk. This is a special programming language designed specifically for working in the *HyperCard* environment. The inclusion of a programming language into a computer application greatly extends the power of the application. Thus, many computer applications now include built-in programming languages. This means that as you learn to use HyperTalk in a *HyperCard* environment, you are also learning about the general idea of using a built-in programming language in other computer applications.

HyperCard Is a Versatile Tool

HyperCard is a versatile and powerful computer tool. It combines the following basic elements of several general-purpose computer tools:

1. Paint capabilities, to use the computer to create graphics. The *HyperCard* system includes a large number of aids for creating animation and other visual effects that enhance the value of graphics in a document.
2. Word-processing capabilities, to use the computer as an aid to writing.
3. Database capabilities, to use the computer as an aid to storing, processing, and retrieving data.
4. Audio capabilities, to use the computer to include sound and audio effects in a document.
5. An interface to other media, such as a CD-ROM or laserdisc player.

In summary, *HyperCard* is designed to facilitate the creation of interactive, nonlinear dialogs between a person and a computer system. The computer system may contain a variety of peripheral devices, such as a CD-ROM and a laserdisc player. As noted previously, an interactive, nonlinear, multimedia document is called a *hypermedia document*, or simply *hypermedia*.

Reading and Writing Hypermedia

It is instructive to examine the parallel between reading and writing text, and reading and writing hypermedia. In both cases you can create documents mainly for your own use, or you can create documents that are designed for others to use. In both cases you can "read" documents created by others. In both cases it takes considerable time and effort to learn to read and write.

Think back to your childhood as you were learning to read and write text. You already knew how to listen and speak. Because reading and writing text are closely linked to listening and speaking, you already had a good start on learning to read and write. Still, learning to read and write text takes a great deal of time and effort. Thus, reading and writing are part of the core curriculum in education at all grade levels and continuing on into college. There, one can even major in fields such as literature, writing, and journalism.

Now consider reading and writing hypermedia. Because you know how to read and write text, you already have a good start on learning to read and write hypermedia. If you have some drawing, design, and other artistic skills, you will find that this helps a great deal in learning to write hypermedia. Probably you have used a word processor and other computer applications. You will find that there is considerable transfer of learning from this type of computer knowledge to learning *HyperCard*. Probably you have used a tape recorder to record and play back your voice and music. *HyperCard* contains facilities for working with audio. Perhaps you have studied computer programming. If so, that will help you a great deal, because HyperTalk shares many characteristics with other computer programming languages.

No matter what your background, it should be evident that learning *HyperCard* and learning to read and write hypermedia are a challenge. You can expect rapid initial progress because the *HyperCard* system was designed to facilitate such rapid initial progress. As you delve deeper into *HyperCard*, you will find it challenging. This will give you the opportunity to study your learning processes and how you deal with a challenging learning environment.

Objects in *HyperCard*

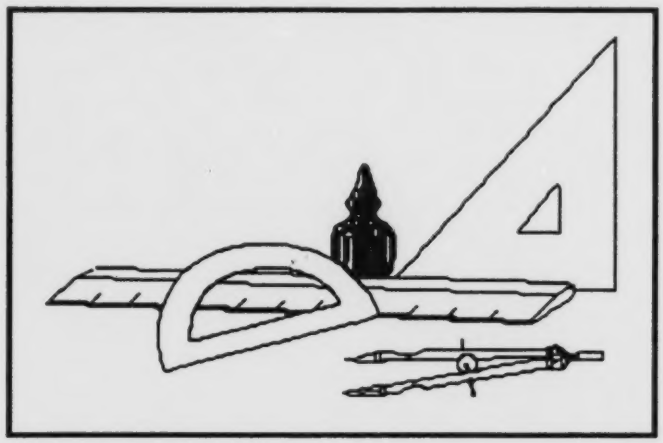
As you learn *HyperCard*, you will learn to create *HyperCard* documents. These documents will be made up of five types of "objects"—cards, buttons, fields, backgrounds, and stacks. Everything you create in *HyperCard* is done by using these objects; everything you see in *HyperCard* documents created by others is accomplished by using these objects.

> *Technical Note: Sometimes you will hear people talking about* HyperCard *as an "object-oriented" programming environment. It would be more accurate to call* HyperCard *an "object-like" programming environment. True object-oriented languages allow you to create new types of objects;* HyperCard *allows you to create only the five kinds of objects previously described. While* HyperCard *includes some of the characteristics of an object-oriented environment, it does not have the full power of an object-oriented programming language.*

Aids to Problem Solving

Humans are very versatile in the range and nature of the types of problems that they address and solve. Over the millennia, humans have developed a wide variety of tools to help in problem solving. Some of these tools augment or extend the capabilities of the physical body. Carpenters' tools, such as the hammer and saw, provide good examples. The telescope and microscope extend the capabilities of eyes. The bicycle and automobile extend the capabilities of legs. A hearing aid augments the capabilities of the ear.

Some of the tools people have created augment or extend the capabilities of the human mind. One can think of reading, writing, and arithmetic as mind tools. Skills in using reading, writing, and arithmetic allow one to address a wide range of problems, many of which are very difficult or impossible to solve without these tools. Reading, writing, and arithmetic offer very broad uses as general-purpose tools because they are applicable across the entire range of human intellectual endeavors.



Any calculating device, be it an abacus, a hand-held calculator, or a general-purpose computer, extends the capabilities of the human mind.

The electronic digital computer in its most general sense is both a tool to aid the physical body and a mind tool. Computerized machinery—including robots—augment and extend physical capabilities. A huge range of computer software, including artificially intelligent programs, extends and augments the user's mental faculties. Having skills in using computers allows one to address a wide range of problems.

The Human Mind

The human mind has tremendous capability and versatility. It is both a storage device and a processing device. As a storage device, it has a huge capacity, but retrieval is somewhat erratic. As a processing device, the brain is good at working with uncertainties. It is a parallel processor, with many millions of neurons simultaneously engaged in a thinking task. However, its accuracy sometimes is not as good as we would like it to be.

Human brains differ substantially in ability. Howard Gardner and Robert Sternberg, among others, have proposed a variety of different ways of categorizing these abilities or "intelligences." For example, Gardner postulates that humans have at least seven different major categories of intelligence, including linguistic, logical-mathematical, musical, and spatial. *HyperCard* is a tool that can help the brain in all of these areas.

Educational Uses of *HyperCard*

HyperCard is now being used throughout education, both at the precollege and college levels. Many educators are now learning *HyperCard* both for its benefits as a personal productivity tool and for use in their classrooms. Thus, there are now many preservice and inservice courses for teachers that focus on the educational uses of *HyperCard*.

The overall field of the instructional use of computers can be divided into three main parts:

1. Computer science, computer programming, and the associated underlying theories.
2. Computer-assisted instruction, computer-managed instruction, and other aspects of using a computer system to help teach and to help students learn.
3. Productivity tools for use by students and teachers, such as word-processing, database, graphics, or gradebook programs.

These three categories are not distinct. A single piece of computer software may fall into all three categories. Educational uses of *HyperCard* fall most frequently into the first and third of these categories. For example, many students use *HyperCard* as a tool to create projects such as reports. However, many teachers use *HyperCard* to create presentations or other materials for their students, and this type of use falls in the second category.

A number of educational software companies use *HyperCard* as the underlying software in their multimedia courseware packages. Thus, it is appropriate to view *HyperCard* as computer application software that covers the full spectrum of instructional uses of computers.

How to Use This Book

This book is designed so that you can use it while working at the computer. You should read a section and immediately try the ideas discussed in that section. Take time to "play around" with and think about new ideas. Don't simply duplicate the examples in the book. Extend them. Explore. Experiment. Create new ideas. Relate the book's examples and ideas to your own knowledge and experiences. If you simply rush from example to example and from section to section, you will quickly become overwhelmed with new ideas and concepts.

The book includes the following aids to assist you in learning:

- The Table of Contents lists every major section in the book.
- There is an Activities section at the end of each chapter.
- There is a References section.
- Three Appendixes appear at the end of the book. Appendix A lists keyboard shortcuts for use in *HyperCard*, Appendix B gives information on creating visual effects, and Appendix C provides an overview of research on problem solving.
- There is a very detailed Index.

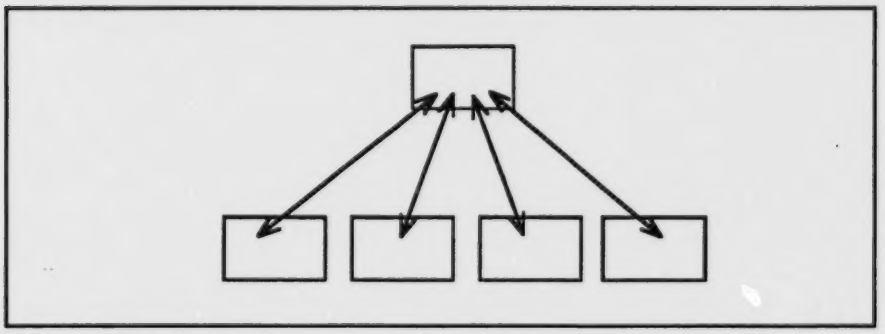
Activities

Activities fall into three major categories. First, there are activities that ask you to write, for example, in a personal journal. Typically these activities ask you to become introspective—to think about your own thinking, feelings, and learning processes. Research suggests that such metacognitive activities are quite useful in improving learning. Second, some activities suggest that you carry out certain *HyperCard* tasks on your own. Such "problem posing" is a key aspect of problem solving. Third, some activities suggest that you create a specific stack and then seek feedback from yourself and others on how well your *HyperCard* stack works. Does it solve the problem and communicate effectively?

The following activities are based on ideas covered so far in this book.

1. Write in your journal about any previous *HyperCard* experiences you have had, and summarize your current level of knowledge and skill. Think of this as a snapshot to be used in a "before-and-after" report on your progress in learning to use *HyperCard* effectively.

2. Examine the five-card stack map given below. Describe two different information storage and retrieval tasks that could be solved by such a stack design. (Hint: As an example, think about the Animals, Mountains, Plants, and Rivers set of index cards discussed previously in this chapter.) The idea represented in the stack map is that the reader starts at the card in the first row and is then directed to one of the other four cards.



3. Write briefly in your journal about your skills in communicating via speaking and listening, writing and reading, composing and performing music, drawing and painting, and so forth. Be aware that there are lots of vehicles for representing and communicating ideas. The computer is a new and powerful aid to representing and communicating ideas.

4. Write in your journal a brief summary of your current knowledge and feelings about problem solving. For example, when you think about problem solving, is your main focus on mathematics? Or do you think

about problems in a number of different domains? You might want to write about some of your major successes and failures in problem solving.

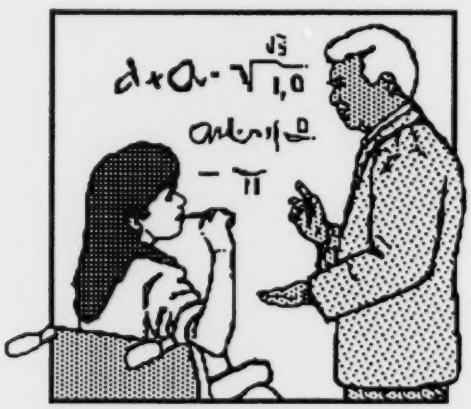
Chapter 2. What Is Problem Solving?

This chapter contains a formal definition of the word *problem*. The focus is on problem solving in a computer environment.

There are many books and a great deal of research literature on problem solving. The following list provides a simple summary of what is known about problem solving.

- Through appropriate study and practice we can all get better at problem solving.
- One way to get better at problem solving is to learn some vocabulary that will help you communicate more precisely (with yourself and others) about problem solving. Use the vocabulary in your talking and writing.
- Another way to get better at problem solving is to learn to use effectively the tools that humans have developed to aid them in solving problems.

Appendix C contains a summary of a number of additional research findings about problem solving.



Problem Situations

Most people use the word *problem* when they are actually talking about a problem-like situation. As you move through a typical day or week, you are apt to encounter a number of problem-like situations. Following is a list of examples. As you read this list, think about whether a computer might be useful in dealing with some of the problem situations.

- You turn on a television news program and are shocked and saddened as you view a program about starving children.
- You have been given a term-paper assignment. You need to find information, organize it, and write the paper—all in the next three weeks.
- You are taking a course in math (or chemistry, or physics, or other subjects.) The homework is to do the odd-numbered problems at the end of the current chapter.
- You have moved to a new location. You need to inform a large number of people that your address and phone number have changed.
- As you use public transportation, you notice that few others are using public transportation and that the traffic and air pollution are terrible.
- You know how much savings you have and can project your income over the next few months. You have a lot of expenses coming up. You need to develop a budget.
- You have been asked to make a presentation to a group of people. The expectation is that your presentation will make effective use of multimedia.

You can easily extend the list. Think of problem situations that involve and concern you. Think of situations where you might be able to take some action that might resolve the problem situation.

As you extend your list of problem situations, begin to look for common features that all or many of the problem situations share. You will probably find that many of the examples have four things in common.

1. There is a description of how things are—the initial situation. Often the initial situation is not carefully specified, perhaps because it is taken for granted or assumed to be obvious.
2. There is an indication of a desired changed situation—some sort of goal to be reached, product to be produced, or task to be accomplished. Sometimes the goal is not clear or there are a number of possible goals.
3. There is an indication that it might be possible to move from the initial situation toward the goal. The world, the nation, the state, the company, the school, you and your colleagues, or some set of resources exist that are relevant to attempting to make this movement.
4. You have an interest (some level of ownership) in the situation. You are willing to expend some thought, time, effort, and/or other

personal resources in moving from the initial situation toward the goal.

Definition of a Formal Problem

Most researchers who study and write about problem solving tend to use nearly the same definition of "problem." In this book we will use a four-part definition of a formal problem. We will say that you have a formal problem—a clearly defined problem—if the following elements exist:

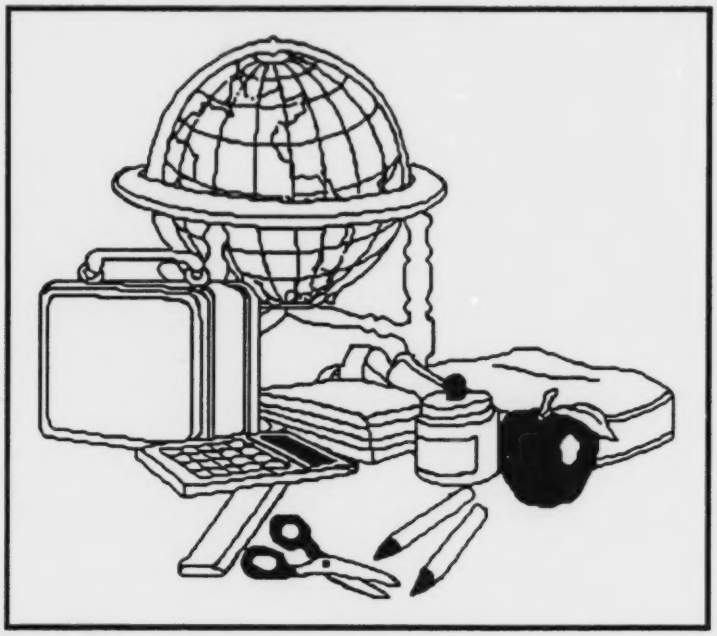
1. **Givens.** There is a clearly defined given initial situation. This is a description of how things are, what is happening, what is known, and so on.

2. **Goal.** There is a clearly defined desired final situation. This is a description of how you want things to be. What do you want to accomplish? How can you tell if the goal has been reached?

3. **Resources.** There is a clearly defined set of resources. What can you and/or others do that will affect the situation? What types of things can you do that might help you achieve the goal? What resources do you have? Resources include skills, knowledge, time, energy, materials, machines (such as computers), money, and so on. In many problem-solving situations, a computer is a very useful resource. Often there are rules and restrictions that limit the resources. For example, in writing a term paper, rules and restrictions might include "Do your own work" and "Don't plagiarize."

4. **Ownership—Importance to you.** For something to be a problem for you, you must have some interest in it. You must accept some ownership. You must want to solve the problem and agree to work on it. Notice that in some cases you may be only mildly interested. In other cases you may be deeply interested. Also, notice that a particular problem-like situation may be a problem for one person (who has ownership) but not for another person (who has no ownership).

Many people find two parts of this definition particularly confusing. First, they tend to think that the resources are a set of directions on how to solve the problem. That is incorrect. The person or persons who are going to solve the problem must try to figure out how to use the resources to solve the problem. There are many problems that cannot be solved within the confines of the available resources. There are many other problems that cannot be solved, no matter what the resources.



Second, very few of the problem situations one encounters are formal, clearly defined problems. Look back at the list of problem situations given in the previous section and/or your own extensions of the list. Pick one of these and analyze how well it meets the four-part definition of a formal problem.

For example, consider the problem situation of turning on the television and being shocked and saddened by a program about starving children. Here is a short analysis of this problem situation.

1. Givens. You are given information about starving children. It is not clear where this is occurring, why this is occurring, how many children are starving, and so on. The program may be a historical documentary on starvation that occurred many years ago, or it might address a continuing pattern of starvation in certain parts of the world. These and many other questions would need to be answered before you could begin to take effective action to alleviate the starvation.

2. Goal. No goal is stated. Some television viewers might decide that the problem is that they are shocked and saddened by what they are viewing—and solve the problem by turning off the television set or flipping to another channel! Others might set a goal of collecting food to be sent to the starving children shown in the television program. Still others might decide on a goal of helping hungry children in their own town.

3. Resources. The problem situation does not contain a discussion of resources. Of course, the television program might be designed to encourage people to donate resources, such as time and money, to help alleviate the starvation.

4. Ownership. Ownership is implied by the words "you are shocked and saddened."

One of the most important ideas in problem solving is obtaining a formal, clearly defined problem. In most problem situations, a great deal of effort is required to move from the poorly defined problem situation to a clearly defined, formal problem.

A Five-Step Plan for Attacking a Problem

There has been a great deal of research on how to solve problems and on how to help people get better at solving problems. A summary of some of the research is given in Appendix C. It should be obvious that there is no easy-to-learn process that will solve every problem. Moreover, methods that are quite useful for solving problems in one area (such as science) may not be very useful in solving problems in another area (such as art, music, or your personal life).

However, there is a five-step plan that can be used as a starting point in attacking almost any problem. It is worth memorizing the steps of this plan and practicing them until you are quite comfortable using them. The five steps in this plan are:

1. Understand the problem. If the "problem" is actually a problem situation that is not clearly defined, work to obtain a formal, clearly defined problem.

2. Devise a plan of action. Figure out what steps you will take to solve the problem. This is a mental process, perhaps carried out with the assistance of writing implements such as a computer or paper and pencil. It typically includes careful thinking about possible direct effects and side effects of carrying out the plan.

3. Carry out the plan.

4. Check to see if the problem is solved. Carefully compare the goal to the outcome that has been achieved. If the goal has not been reached, you will probably need to return to step 1, 2, or 3.

5. Check to see if new problems have been created. Often the process of solving a problem creates new problems. The problem solver has a responsibility to identify and address these new problems.

The five-step plan is presented in a linear order. In actuality, people jump around in these steps, moving forward and backward in the list as

they gain increased understanding of a problem and the difficulties it presents. Notice that in step 2, one thinks about a plan. A good thinker will imagine carrying out a plan and the consequences of carrying the plan. Thus, step 5 may be addressed (mentally) before step 3 occurs.

Domain Specificity

Suppose you are very good at playing chess. That does not mean that you will be good at solving interpersonal problems. Chess and interpersonal problems are two quite distinct domains.

One of the key ideas in problem solving is *domain specificity*. This is a fancy way of saying that you must know a great deal about a specific problem area in order to solve problems in that area. Suppose you are going to use *HyperCard* to solve an art problem. You must know about two domains—*HyperCard* and art.

HyperCard is a general-purpose tool that is useful in addressing problems in many different domains. However, a knowledge of *HyperCard* is not a substitute for a knowledge of the domain of the problem to be solved. Typically, solving a problem requires a combination of a knowledge of the problem-solving tools available for solving the problem and a domain-specific knowledge in the problem area.

Transfer of Learning

It is not possible to provide examples of how to use *HyperCard* in every domain or to list the full range of problems that can exist in any specific domain. Thus, you—the reader—must assume responsibility for connecting your growing knowledge about *HyperCard* to your current knowledge about many different domains and the problem-solving tools available in these domains. This is a task involving the *transfer of learning*.

There has been a great deal of research on transfer of learning. Some of the key ideas are summarized by the High Road/Low Road theory of Salomon and Perkins.

Low Road transfer occurs through development of *automaticity*. Probably you have fully internalized the rule "Look both ways before crossing the street." You apply this rule automatically and without conscious thought whenever approaching a street—even when crossing a one-way street! In Low Road transfer, your brain has stored a stimulus pattern and an automatic response that is applied whenever the pattern occurs.

Generally speaking, it takes a great deal of practice to achieve automaticity. Perhaps you are an accomplished touch typist. If so, you can type at a keyboard far faster than you can write by hand. This skill

transfers quickly and easily among different keyboards. But it took a great deal of time and effort for this Low Road transfer to occur.

High Road transfer involves careful, conscious thought. One way to solve a complex problem is to break it into two or more smaller problems. This strategy can be memorized, and it is applicable in lots of different situations. Through extensive practice you can develop skill in using the strategy of breaking big problems into smaller components. However, it takes careful thinking to know if this is a useful strategy in a particular situation, and it takes careful thinking to subdivide a complex problem in a useful manner.

This book contains a number of strategies (such as breaking a big problem into smaller parts or using the five-step plan for attacking a problem) that are useful in *HyperCard* and in many other problem-solving domains. When you encounter one of these strategies, add it to your working knowledge. Practice using it throughout the day with the many different types of problems you encounter. Gradually you will find that High Road transfer is occurring and that your overall skill at solving problems is improving.

Activities

1. Write down two or three examples of clearly defined problems that are solvable but that you feel a computer cannot solve. What are the common characteristics of these problems that make them unsuitable for being solved by a computer? (Note: Here and in the remainder of this book, the "write down" types of activities can be considered as journal writing activities and/or as assignments to be handed in to a teacher. In either case, spend time thinking about the activity—think about possible things you might learn by doing the activity. Do metacognition about your thought processes in doing the writing.)

2. Write down two or three examples of clearly defined problems that are solvable and that a computer can solve. Select problems that fully utilize a computer's capabilities—that require a reasonably powerful computer. What are common characteristics of problems that can be solved by computer but that require a reasonably powerful computer? (Think about what might be meant by "a reasonably powerful computer." What makes a computer powerful?)

3. Write down a problem situation that is not a clearly defined problem. Develop three different clearly defined problems from this problem situation.

4. Give some examples from you own life and/or from media (such as television ads) that illustrate domain specificity. (Example: a very

successful athlete giving an opinion on a non-athletic problem in order to convince you to buy a product unrelated to athletics.)

5. Give some personal examples of Low Road transfer. Give some personal examples of High Road transfer. In each case, clearly identify the types of problems being addressed.

6. Analyze your greatest strengths and your greatest weaknesses as a problem solver. For example, do you have persistence? Are you good at formulating goals? Are you good at identifying resources? (This is a metacognitive, journal-writing activity.)

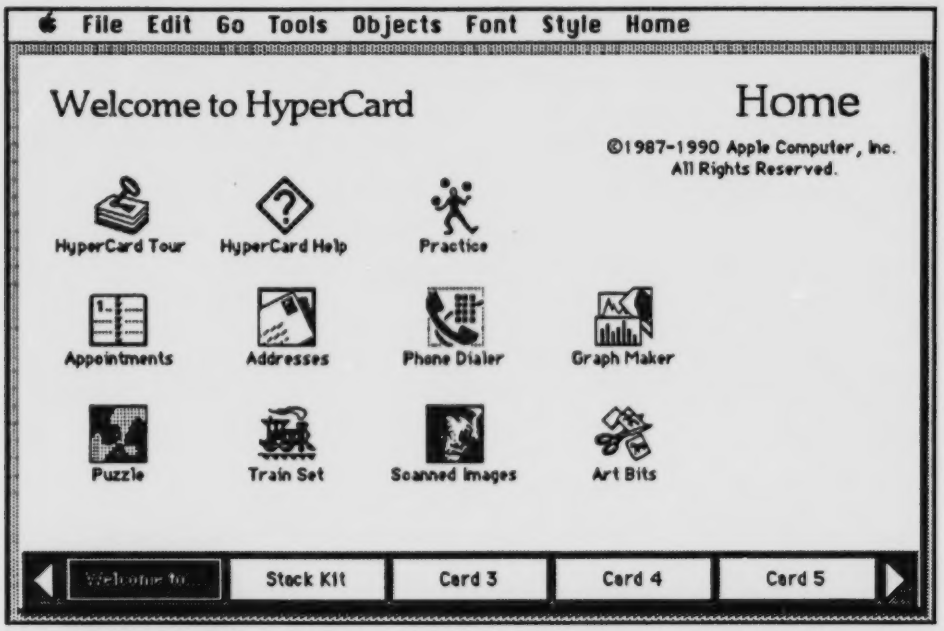
Chapter 3. Getting Started in *HyperCard*

This is a hands-on chapter. The assumption is that you are seated at a Macintosh computer that has the *HyperCard* software properly installed. The goal in this chapter is to explore some of *HyperCard*'s built-in features. Some of these features are discussed from a communications and problem-solving point of view.

A First Look at *HyperCard*

To start *HyperCard*, you can either click on the *HyperCard* icon and choose **Open** from the **File** menu or you can double click on the *HyperCard* icon. This starts *HyperCard* with the Home Card showing. If you are using Version 1.2.5, your screen should look something like this:

If you are using Version 2.0 or above your screen may look something like this:



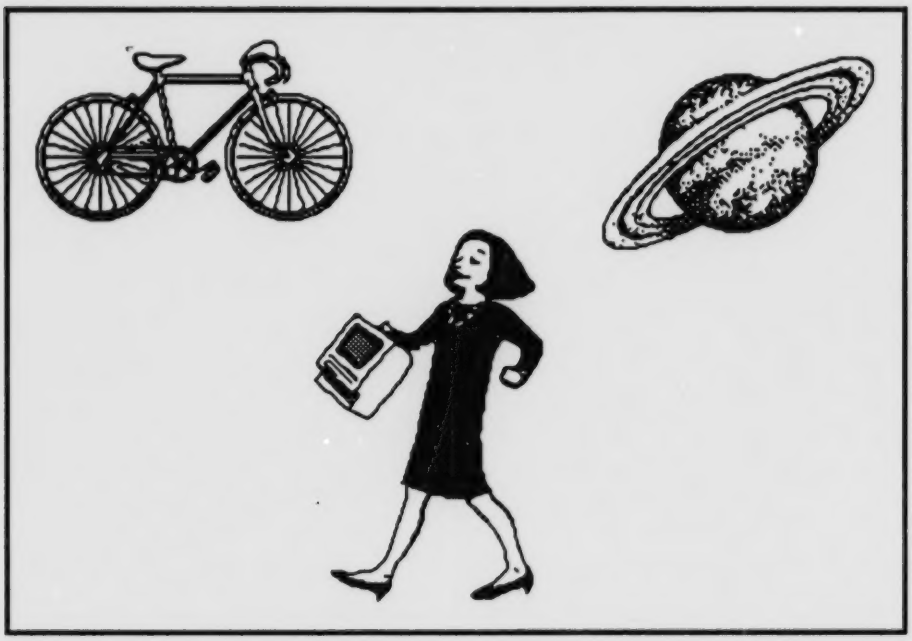
Technical Note: In this book, screens will be shown as they appear on the "traditional" 9-inch Macintosh screen. That is, the Menu bar will appear along the top edge of the card. If you have a larger screen, the Menu bar will be above the card.

Getting Familiar With *HyperCard*

Upon starting up most versions of *HyperCard*, the user sees a Home Card that contains a number of buttons. In the two examples just shown, there are rows of pictures—called icons—that are buttons. Clicking on one of the buttons opens a particular stack that accompanies *HyperCard*.

Technical Note: It is assumed that you have previously used a Macintosh with a pointing device such as a mouse or a trackball. This book uses the term "mouse" to refer to any type of pointing device.

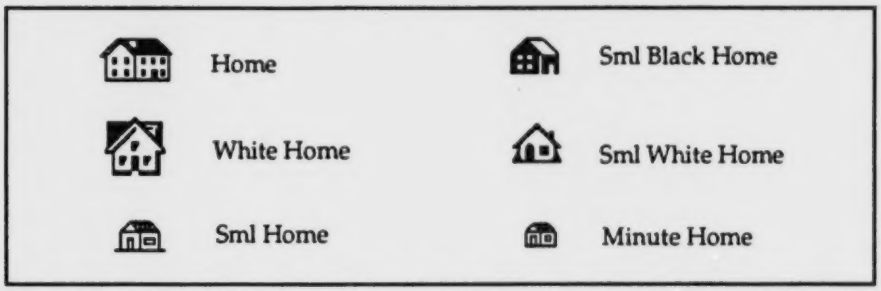
These stacks can serve as valuable resources. For example, you may find that an appointment book stack exactly fits your needs for a computerized appointment book. You may find that some of the clip art is exactly what you need for a particular project. Details on how to use the clip art are given later in this book. The following graphics came from clip art that accompanies one of the versions of *HyperCard*.



You will want to explore a number of different stacks. Select one of the buttons that appear on your Home Card. If an icon appears to lead to a stack containing some art, you might want to look at the stack to see what clip art is available. You might want to take a look to see if there are useful ideas for your work. Maybe an icon just looks like fun. Select it and see what you can find.

> *It is possible that your copy of* HyperCard *does not include the stacks indicated on the Home Card. If* HyperCard *asks you to find a particular stack, check in any likely folders on your hard drive that might contain* HyperCard *material. Sometimes people remove these sample stacks in order to save computer memory. You may also have a version of* HyperCard *that has a very limited number of sample stacks.*

As you use a stack, think about how well it is written. Is its purpose clearly explained? Are the directions clear? Can you always tell what to do next? Can you always tell how to get back to the Home Card so that you can explore a different stack? It is common to use a button that looks like a house to send the user back to the Home Card, which is the first card of the Home stack. Thus, if you become confused when exploring a particular stack, look for a house and click on it.



Exploration is a trial-and-error process, a learn-by-doing process. You need have no fear that you will damage the computer. If you get thoroughly confused by a stack you are exploring and cannot find your way back to the Home Card, you can always look in the **File** menu for the **Quit *HyperCard*** choice. You can quit *HyperCard* and start all over again.

Explore several of the stacks that can be opened from the Home Card of the *HyperCard* system you are using. Develop confidence that you can open a stack, use it, and return to the Home Card. That is, learn to "navigate in hyperspace." Begin to internalize the idea that a stack is a set of cards, typically connected in a nonsequential manner. Develop a mental model of where you are in a stack and what you need to do to move from location to location.

After you become comfortable with opening and exploring *HyperCard* stacks, use the **Quit *HyperCard*** option in the **File** menu to leave *HyperCard*. Then start *HyperCard* again (if possible, without referring to this book or your notes), open another stack, explore it, return to the Home Card, and quit *HyperCard* again. It is important that you become thoroughly comfortable in accomplishing this sequence of tasks.

Think about the general idea of turning on a computer system, selecting and starting a particular application, using the application to accomplish a desired task, shutting down the application, and shutting down the computer system. Now think about Low Road and High Road transfer and your own learning. If you want to achieve High Road transfer in the multistep, "use-a-computer-application" process just described, carefully and consciously think about these steps as you use a variety of computers and a variety of computer applications. Look for other similar or nearly similar situations, such as in turning on a stereo system and selecting a particular component such as the tape deck.

Problem Solving—A Thought Experiment

You have now seen how easy it is for a novice computer user to use a stack someone else has created. Let's try a "thought experiment." Imagine that a huge number of stacks has been created, one for each problem a

computer can solve. A gigantic Home Card has been created that contains an appropriate icon for each of these stacks. Then a novice computer user could solve any problem a computer can solve by merely selecting the appropriate icon, clicking on it, and following the directions the computer provides. Even a young child could gain immense power as a problem solver. What are the educational implications of providing all students with access to such a computer system?

To a certain extent, the thought experiment represents a piece of reality. Many billions of dollars have been spent developing computer programs designed to solve specific categories of problems. There are tens of thousands of computer programs available commercially. Thus, computer programs exist that solve or help to solve many commonly occurring problems.

Most people spend a great deal of their creative problem-solving efforts "reinventing the wheel." One of the most important ideas in problem solving is to build on previous work done by yourself and others—don't reinvent the wheel. Computers bring a new dimension to problem solving and to communication specifically because they are an effective way to store certain types of previous work. If a person figures out how to use a computer to solve a problem, this knowledge can be stored in a form that others can also use. Often this use is much easier than starting from scratch to figure out how to solve the problem.

Why Learn to Create Stacks?

The previous section might be considered to be an argument that most people have no need to learn to create *HyperCard* stacks. However, that is incorrect.

First, many of the problems *you* encounter are quite specific to you. Nobody else has encountered exactly the same problem, recognized that it can be solved using a computer, developed the necessary software, and made it available.

Second, you can get better at creatively dealing with new problem situations through study and practice. It is important to practice on problems that are of a manageable size and that have been designed to help promote learning, even though millions of people may have already solved exactly the same problems when they were students.

Third, it is fun to be creative and to create. *HyperCard* provides an excellent environment for expanding your creative intellectual communication and problem-solving skills.

Fourth, and most important, *HyperCard* operates in a communication environment. It involves reading and writing along with graphics, animation, and sound. It is a new medium for nonsequential

communication. One learns to read and write hypermedia by working with *HyperCard*.

Beginning in the next chapter, you will learn how to create your own stacks—that is, you will learn to write hypermedia in a *HyperCard* environment.

Activities

1 Have you tried the *HyperCard* ideas discussed in this chapter? That is, have you used a hands-on approach to studying the material in the chapter? If the answer is "yes," go on to the next activity. If the answer is "no," carry on a conversation with yourself about why. Your "conversation" may be in the form of a journal-writing activity.

2. In this chapter you have used two kinds of menus. You have used a pull-down menu named **File**, and you have used a menu consisting of a number of buttons (the Home Card). Each provides a certain type of interface with the user. Discuss the advantages and disadvantages of each of these two designs. When might one approach be preferred to the other? Which seems best to you? Why?

3. This chapter mentioned the idea of navigating in hyperspace. What is hyperspace? What does it mean to "navigate in hyperspace"? Discuss the use of this metaphor (the words *hyperspace* and *navigate*). Is this metaphor useful to you, or does it hinder more than it helps?

4. Think about some domain you know quite well, and think about the domain-specific knowledge and vocabulary needed to express the ideas and problems in that domain. Discuss the possibility of having stacks for solving the problems, with the stacks designed to be used by people who do not know the vocabulary of the domain and who do not have the domain-specific knowledge.

5. A hand-held calculator contains buttons for carrying out four mathematical operations: adding, subtracting, multiplying, and dividing. Discuss similarities and differences between this concept of "buttons" and the buttons used in *HyperCard*. If it seems appropriate to you, make use of the terms *domain specificity, domain independence,* and *transfer of learning* in this discussion.

6. Note that the previous activity is a High Road transfer activity. Certain tasks can be accomplished by pushing buttons. There are lots of different situations where pushing a button is an appropriate thing to do to help solve a problem. Make a list of "button-pushing" types of problem situations, including using an elevator. Discuss common characteristics of these situations. In these situations, is it always obvious what buttons to push and what the outcome will be? Is it always easy to tell if you have pushed the right button and that you

have pushed hard enough? Is it always possible to correct an error, such as pushing the wrong button?

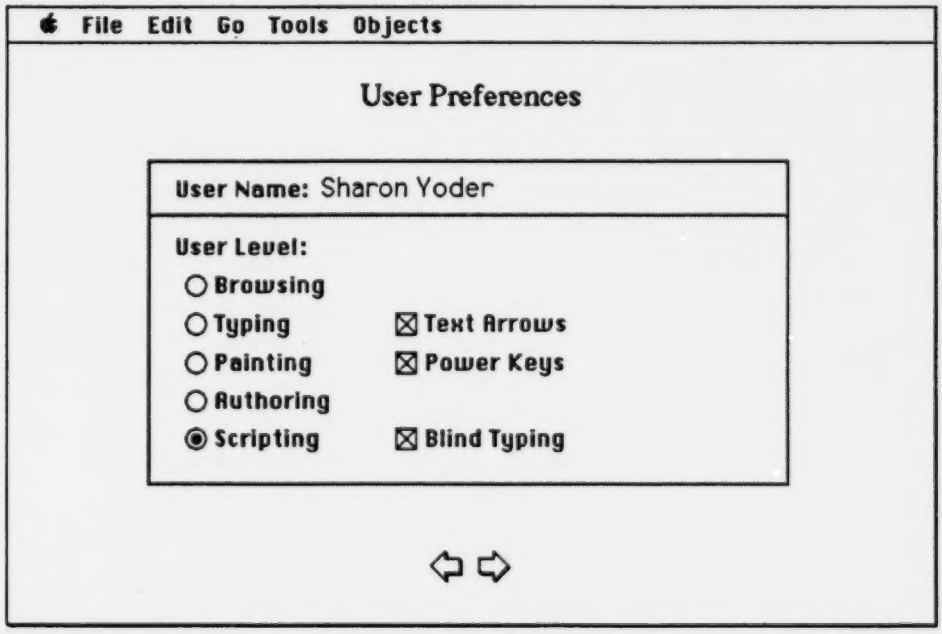
Chapter 4. Making Cards and Stacks

In the previous chapter you saw how easy it is to use stacks created by others. In this chapter, you will learn to create your own stacks. Keep in mind that you are creating stacks to be used by you and by others. You are designing an interactive message that uses a computer system. You are learning to communicate in hypermedia. In addition, you are learning to write computer programs!

Communication in *HyperCard* is a two-phase operation. First, you communicate with the *HyperCard* application as you create ("write") a stack. Then, you or someone else uses the stack and the *HyperCard* application software to use ("read") your stack. That is, your stack communicates with the person using it.

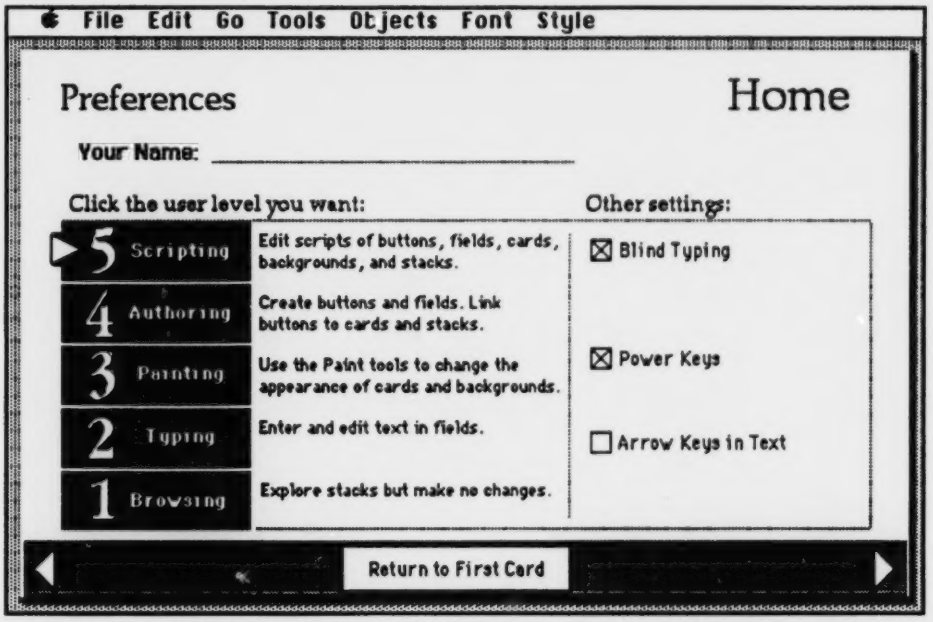
Setting the User Level

To begin a new *HyperCard* project, you first need to create a new stack. Before creating a stack, however, you need to be sure that your User Level is set correctly. If you are using Version 1.2.5, click on the Left Arrow at the bottom center of the Home Card. You will then see the following card.



Click **Scripting** and then return to the Home Card by clicking on the Right Arrow.

If you are using Version 2.0, select **Prev** from the **Go** menu. Set the User Level to 5 by clicking on **Scripting**.


File Edit Go Tools Objects Font Style
Preferences
Home
Your Name:
Click the user level you want:
Other settings:
5 Scripting
Edit scripts of buttons, fields, cards, backgrounds, and stacks.
Blind Typing
4 Authoring
Create buttons and fields. Link buttons to cards and stacks.
3 Painting
Use the Paint tools to change the appearance of cards and backgrounds.
Power Keys
2 Typing
Enter and edit text in fields.
Arrow Keys in Text
1 Browsing
Explore stacks but make no changes.
Return to First Card

Then return to the Home Card by clicking on the Right Arrow. Note that in either version of *HyperCard*, different boxes on the right side of the Preferences card may be checked.

Technical Note: If ***Painting, Authoring,*** *and* ***Scripting*** *do not appear, select* ***Message*** *from the* ***Go*** *menu. When the Message box appears, type the word "magic." Close the Message box by clicking in the small square in its upper left corner. Now you are ready to set the User Level. You must be on the Preferences card for this to work correctly.*

Creating a New Stack

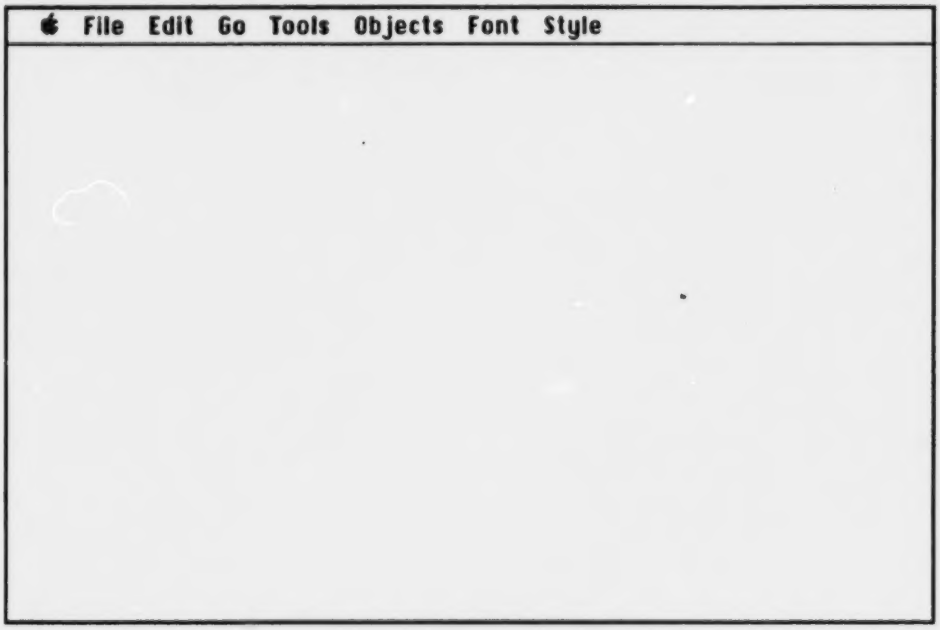
Look at the **File** menu. It contains a number of menu choices, such as **New Stack** and **Open Stack**. Select **New Stack**. First the *HyperCard* system will ask you to name the new stack. Below the box where you type the name of your new stack is a check box that allows you to decide whether to use the current background in your new stack. You do *not* want to use the current background. You want a *new* background.

Technical Note: In Version 1.2.5, you must remove *the check from that box by clicking on the "X." If the box is checked, this will cause* HyperCard *to copy the background from the Home stack and use it to build the new stack.*

Name your stack and click on **New**. You should then have a new stack that contains a single card with a blank background. That is, your screen should contain nothing except the Menu bar at the top of the screen.

Technical Note: If you are using a larger Macintosh screen, you may also see a bar with the name of your stack below the Menu bar.

Your new stack will look like this—a single blank card.



Cards

We have previously used the analogy that a card in *HyperCard* is like an index card. You can write and draw on a *HyperCard* card, just as you can write and draw on an index card. To make this model of *HyperCard* cards more accurate, think of a card as being made of clear plastic or glass. You can write directly on the card (glass) and you can stick things on the card (glass), but at the same time you can see through the card (glass).

Why should you think of the card as transparent? Because each card in *HyperCard* has a *background*. You can see the background through the card, so whatever is put on the background is visible through the card (unless you put something onto the card on top of an object on the background). In the first part of this book it is assumed that the background you are working with is blank. In Chapter 8 you will learn to create and edit backgrounds that are not blank.

Creating Additional Cards

HyperCard provides the user with a number of long menus. Cards can be added to a stack by appropriate use of menu choices. Eventually you will become familiar with many of the choices under each menu. You will gradually memorize the ones you use most frequently. Also, you will notice that many of the menu choices have keystroke equivalents. For example, under the **Edit** menu, the keystroke for **New Card** is ⌘-N (Command-N). Appendix A contains a summary of the keystroke commands applicable to the material in this book.

Spend some time examining the choices in each of the seven menus *HyperCard* provides. You will notice that some of the menu choices are similar to other Macintosh applications you have used. For example, there is a variety of **Print** choices in the **File** menu. You will also notice that there are many boldfaced choices in the menus, and many choices that are not boldfaced. The boldfaced choices are options available for use at the particular time you are viewing the menu.

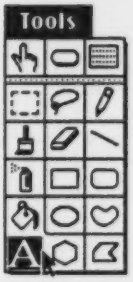
Your stack already contains one card. You can create another new card by choosing **New Card** from the **Edit** menu. If you prefer, you can press ⌘-N (Command-N) to create a new card. Nothing on your screen changes. How do you know you actually created a new card? (This is somewhat analogous to pushing a button but not knowing whether you pushed hard enough. Some people will chose **New Card** a number of times, expecting to see that something has happened. They will end up with a number of new cards, but still will not have any visual feedback that new cards have been created by the computer system.)

To see that a new card has been created, go to the **Objects** menu and choose **Card Info**. A dialog box appears that gives you information about your card. Most likely it says that you are on card 2 of 2 cards. Your stack now consists of two identical cards. You can select **Prev** or **First** from the **Go** menu to get to the first card and then select **Next** or **Last** to get back to the second card. When you are on the first card, the **Card Info** will indicate that you are on card 1 of 2; when you are on the second card, the **Card Info** will indicate that you are on card 2 of 2. Practice moving between the two cards, making use of **Prev**, **Next**, **First**, and **Last**.

At this point you should give each of your cards a name. To name a card, select **Card Info** from the **Objects** menu and then type the name in the dialog box. You should always choose something meaningful to use for the name. For now, call your cards "Demo1" and "Demo2." You should get in the habit of *always* naming your cards as you create them. Using names for all your cards will make your future work with HyperTalk much easier.

Using the Tool Window

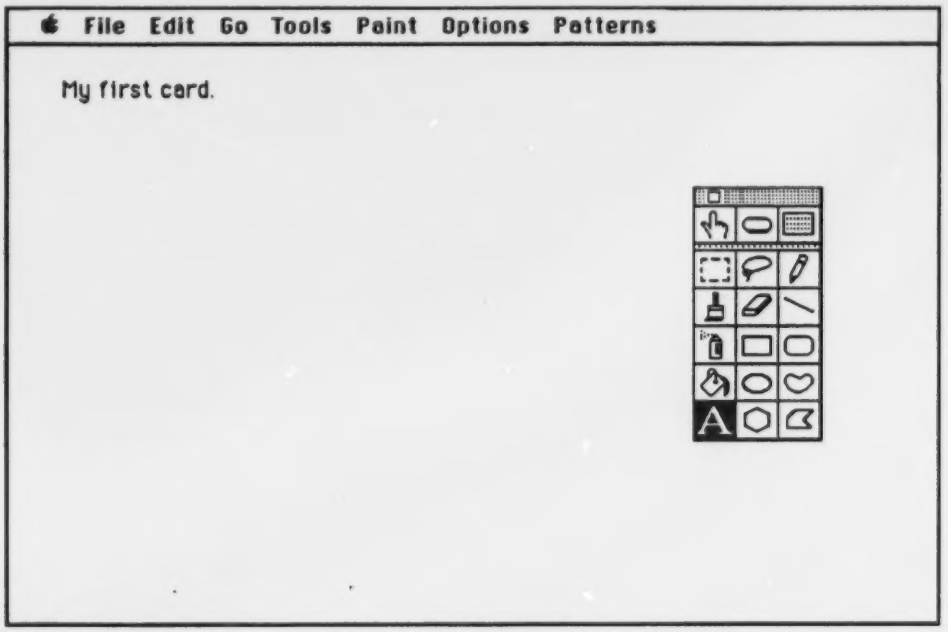
To tell the difference between the two cards you have created, you can draw something on each one. Go to the first card, the one you have named Demo1. Next, move the mouse pointer to **Tools** on the Menu bar. Press down the mouse button and "drag" the mouse down and to the side. You are now using a "tear-off" menu. When you release the mouse button, the Tool window should be visible on your screen. Click on the letter "A" in the Tool window. This selects the **Text** tool.



Technical Note: The process of moving the mouse pointer to a desired location, pressing and holding down the mouse button, and moving the mouse is called "click and drag." The click-and-drag operation ends when the mouse button is released.

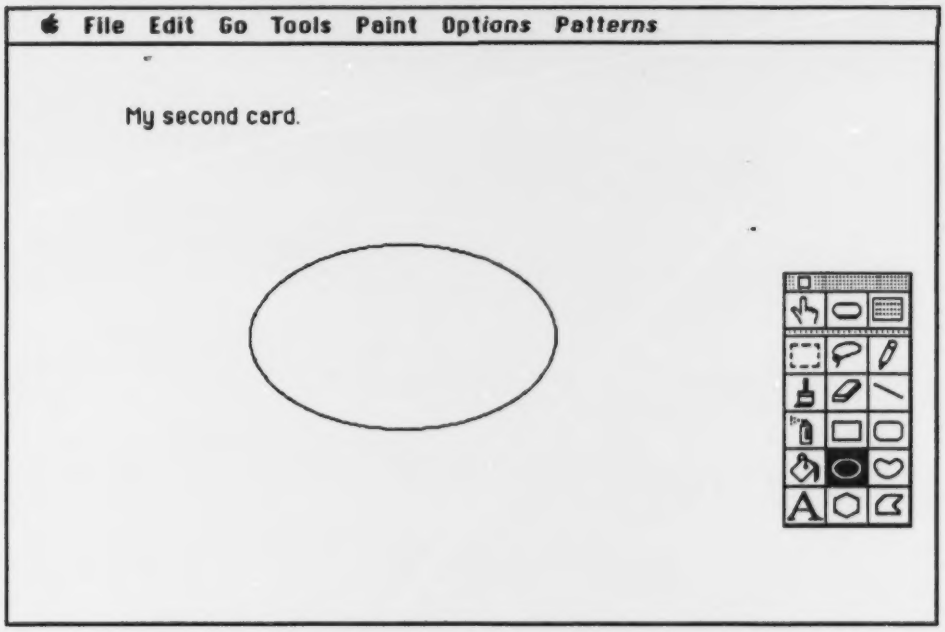
Close the Tool window by clicking in its small square close box near the top left corner. Practice tearing off the Tool window and placing it at different locations on the screen. As you hold the mouse button down and drag the window to the location where you want the Tool window to be placed, you'll see a flashing rectangular outline move across the screen. When you release the mouse button, the Tool window will appear at the location of the rectangular outline.

After the **Text** tool is selected, you can use the mouse to move the pointer to any desired location on the card, and then use the keyboard to type text. Put the words "My first card." on the left side of the card Demo1. The "text" you put on the card with the **Text** tool is actually "painted" onto the card. That is, the letters behave like drawings, not like word-processed text.



Notice that the menu along the top of the screen changed when you selected the **Text** tool. Click on the **Browse** tool (the pointing hand) in the Tool window to get back to the original menu. Then use the **Go** menu to go to your second card. Notice that the Tool window is still available exactly where you placed it as you were working on the first card. Use the **Text** tool to put the words "My second card." on this card.

Next, draw an oval on the left side of the card. To do this, click on the **Oval** tool in the Tool window. Then use the mouse to position the pointer on the card. Hold the mouse button down and drag the mouse a short distance in a diagonal direction. When you release the mouse button, you will see that you have drawn an oval.



Now you can easily tell which card you are on as you move through your new stack. Note that if you continue to select **Next** or **Prev** from the **Go** menu, you will cycle through your two cards repeatedly. The "next" card after the second card is the first card. The "previous" card from the first card is the second card.

The Paint Tools

It may be that you are already familiar with the Paint tools in the bottom five rows of the Tool window. (If not, do not be concerned. The next chapter covers the Paint tools in considerable detail.) These are similar to Paint tools available in a variety of Macintosh applications. Try experimenting with some of these tools. For example, select the **Eraser** tool from the middle of the third row. The pointer becomes an eraser that is used in a click-and-drag movement to erase painted text or graphics created by using the Tool window tools.

Some people feel quite comfortable learning in a trial-and-error mode. Others are much more comfortable when given precise instructions about what to do and what to learn. Think about your own learning style. Try deviating from your usual pattern. For example, if you feel uncomfortable in a trial-and-error situation, experiment with the Paint tools in a trial-and-error mode.

Trial and error is a very powerful way to solve a problem—especially if trials are easy to carry out and there are few or no penalties for an error. In the *HyperCard* system you can experiment to your heart's desire. You

and the computer system provide the feedback. You can usually "see" if you are producing the results you expect.

Copying and Pasting a Card

Often you will want to create a number of cards that are nearly the same. This is easily accomplished by copying and pasting a card. Go to your Demo2 card and select **Copy Card** from the **Edit** menu. Then select **Paste Card** from the **Edit** menu. This process creates a new card that is identical to the card that was copied—it even has the same name! The new card is inserted into your stack immediately *after* the card that shows on the screen when you do the Paste. After the pasting operation, you are on the newly pasted card. Select **Card Info** from the **Objects** menu and name the new card Demo3. Use one of the tools from the Tool window to paint an identification on this card.

Next, experiment with copying and pasting cards. For example, copy your third card and paste a copy into your stack immediately after the current first card.

Deleting a Card and Cutting a Card

The **Edit** menu contains a **Delete Card** option. When you select this option, the card you are currently viewing is deleted. For example, suppose you have a stack consisting of six cards and you delete the third card. The stack will then have five cards. The card that used to be the number 4 of 6 cards will have become the number 3 of 5 cards.

The **Edit** menu also contains a **Cut Card** option. When you select this option, the card you are currently viewing is cut (removed) from the stack.

Delete Card and **Cut Card** both remove a card from the stack. The difference is that after using **Cut Card**, the **Paste Card** option will paste in the card that has most recently been cut. If a card is removed from the stack by using **Delete Card**, that card cannot be pasted back into the stack by using **Paste Card**.

Notice that in either case, the **Edit** menu contains an **Undo** choice. Experiment with its use. It is a major help in learning and in problem solving by trial and error!

Navigation in a Stack

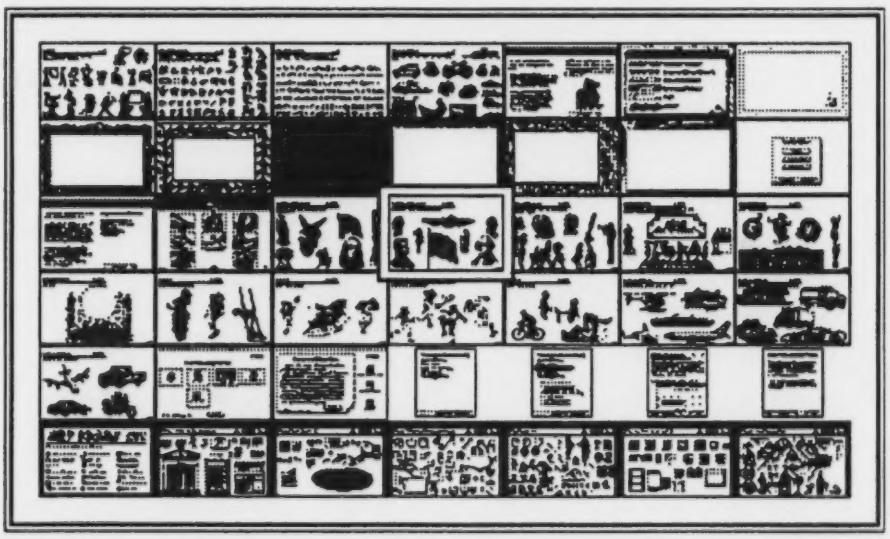
By now you have probably created quite a number of cards. The *HyperCard* system thinks of these as being in a linear order—first card, second card, third card, and so forth. More than likely, the names you have given to the cards do not correspond to this order. Perhaps your stack is still more confusing because the identifying information you have

placed on the cards by using tools from the Tool window no longer makes sense to you and/or is not in "logical" order.

Congratulations! You are well on the way to being lost in hyperspace! You have demonstrated how easy it is to create a stack that confuses even you, the creator. Imagine the confusion such a stack creates when someone else attempts to use it.

When you write in a natural language, such as English, a lot of rules help make the writing readable by others. Sentences start with capital letters, material is organized into paragraphs, and paragraphs are organized in sections in a logical fashion to convey the message of the document. Eventually you will learn rules that will help you organize your stack in a manner that lets you and other users of the stack use it effectively. For the time being, be aware that by using the **Next** option in the **Go** menu, or by using the ⌘-3 (Command-3) keystroke, you can move through your cards in a linear manner. You can sequentially view each card.

In addition, notice the **Recent** choice in the **Go** menu. It produces a display containing miniature representations of the cards you have visited most recently. Only one representation is given for each card, no matter how many times you have visited the card. The following **Recent** screen resulted from exploring a number of *HyperCard* stacks.



Each of the miniature representations behaves like a button. Clicking on a miniature representation moves you to that card.

Suppose you create a particular card in your stack, and you want to take a look at some cards in other stacks. For example, suppose you want to examine a stack you previously examined when you were playing with the Home Card. Use the **Home** choice in the **Go** menu to get to the Home

Card. Browse through the stack of your choice. Then use the **Recent** choice in the **Go** menu. Click on the miniature representation of the card you just created in your stack, and you will be back at that card.

Think of a stack as a document—as an article or book. It is easy to be reading in one document, switch to a different document, and then return to the original document. There is a rapidly growing library of *HyperCard* stacks. If there is a large library of such stacks on the computer system you are using, this is just like having easy computer access to a large collection of library materials.

Saving a Stack

HyperCard is designed so that it automatically saves your stack when you quit *HyperCard*. In addition, it automatically saves the card you are looking at each time you move to another card. These two features take care of the types of problems that occur when a beginner fails to save a document that has required a lot of work. For example, if the power goes off, you will lose only the work that has been done on the card you are working on at that time.

The Idea of a Primitive

This chapter contains a number of basic, relatively easy tasks. It is easy to create a new stack, add cards to the stack, delete cards, copy and paste cards, name cards, and select a paint tool.

In the field of problem solving, a *primitive* is a task that is easily accomplished or a problem that is easily solved. A primitive is a building block useful in solving more complex problems. One way to think about a computer programming language is that it consists of a number of primitives.

There are hundreds of different programming languages. Each programming language contains its own unique set of built-in primitives. Languages such as BASIC and FORTRAN contain primitives especially useful in solving math and science problems. A language such as COBOL that is designed to help solve business problems contains a number of primitives that solve fundamental problems in business. Similarly, Logo is a language that was originally designed for children to use to explore mathematical ideas with a turtle cursor. *HyperCard* contains a large number of primitives useful for creating hypermedia documents.

Learning *HyperCard* involves much more than just memorizing the primitives. It requires learning to think—to solve problems and to communicate by using the primitives. That is the essence of learning any programming language. That is why this book contains so much focus on problem solving and on learning.

Learning *HyperCard*

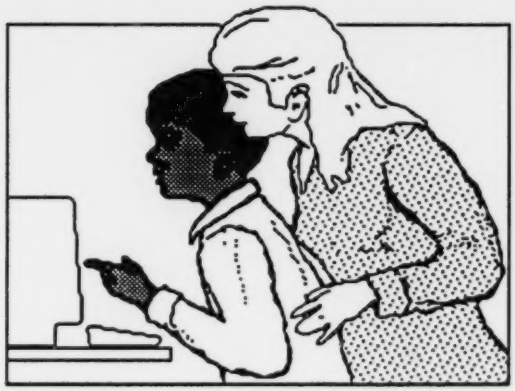
This chapter contains a large amount of information. People vary tremendously in how long they take to learn this amount of information, how well they can apply the information they are learning, and how well they remember it over time.

You might be saying to yourself: "I can learn one new thing, and I can learn two new things. Indeed, I can learn five new things. But, already, the number of new things you are expecting me to learn is overwhelming. There is no way that I can learn this many new things and use them to solve problems. I am overwhelmed."

There is a very good chance that sooner or later as you work to learn *HyperCard* you will be overwhelmed. This happens to almost everybody. One of the main difficulties is separating what is really important from what is less important. Many learners find it is helpful to "chunk" the information to be learned. Focus on the "forest" rather than on the "trees."

For example, one way to think about the current chapter is that it contains details on creating a stack, adding cards, naming cards, copying cards, pasting cards, cutting cards, deleting cards, navigating, saving a stack, and so forth. Or you might think of this chapter as focusing on the use of the menus to create and edit a stack of cards. This "chunk" covers the essence of the chapter. The details are sufficiently obvious or so easily relearned that it is not necessary to explicitly memorize them. For many learners, it is "obvious" that if you are going to create and edit a stack of cards, you need to be able to add cards, delete cards, move cards around, and draw on the cards. If it is obvious enough to you, you can reconstruct how to accomplish these tasks without explicitly memorizing all of the details. Eventually, you will memorize the details of tasks that you need to perform frequently.

It may help you to know that lots of people, including many young children, have learned *HyperCard*. It may help you to know that in the past you learned to speak and listen, read and write. These are far more complex achievements than learning *HyperCard* at a level covered in this book.



Studying *HyperCard* can be viewed as a unique learning opportunity. Do some metacognition, some thinking about what it feels like to be faced by a complex learning situation. Use it as an opportunity to examine your learning strategies.

Bugs and Debugging

Computer programmers call an error in a computer program a *bug*. This goes back to a time when a computer failed to function properly because a real insect (a "bug") got stuck in the circuitry. The process of removing errors from a program (removing bugs) is called *debugging*.

The metaphor of finding bugs and doing debugging, or detecting and correcting errors in one's work, is applicable both in computer programming and in many other endeavors. It is an important metaphor applicable in all problem-solving tasks. It is very easy to learn and practice this metaphor when programming a computer, and this is one of the unique aspects of the computer programming environment. However, it is important to ask if the knowledge and skills you gain for debugging in a computer programming environment transfer to other areas.

Very young children are not afraid of making mistakes. They learn through trial and error as a result of appropriate feedback. It is natural to make mistakes. Mistakes are an essential part of the learning process. The nature of children's learning experiences gradually shapes how they learn. Some young people eventually "learn" that it is better not to try than to face the consequences of failure. This can be a major handicap to learning. Writing and debugging *HyperCard* stacks is one way to become more comfortable with making and correcting errors.

Learning in a *HyperCard* Environment

Have you ever asked yourself what it means to "know" something or to have learned something? Think of some small part of an academic area

you know quite well. How do you know that you know it? How can someone else tell that you know it?

One answer is that you can use the knowledge. You can demonstrate to yourself and others that you can use the knowledge for personal purposes, for accomplishing specific tasks, for helping other people, and so on. *Learning* and *doing*, or making use of one's knowledge, are closely intertwined. Indeed, we know that most students learn best by *doing*. For them, a good instructional environment is one that includes substantial opportunity to practice using what they are learning.

However, an equally important aspect of a good learning environment is feedback. The feedback can come from the learner. For example, consider a student who has written a poem about a cat. The student might say, "I really feel good about this poem. When I read it to myself, it brings a smile to my face. However, the rhyme scheme still doesn't sound quite right, and I want to say more about the cat."

Of course, the student could also get feedback from a teacher or from fellow classmates. Such external feedback is often very important. A teacher needs to be skilled in providing appropriate feedback to students.

A computer environment adds an important new dimension to learning. A computer can help provide feedback to the learner. As you learn to develop *HyperCard* stacks, you will get feedback from yourself, from other people, and from the computer you are programming. Programming means writing instructions to tell a computer what to do. As you attempt to write a program, you will have in mind what task you want the computer to accomplish. You will observe what the computer actually does (and/or fails to do). You and the computer together will provide you with feedback.

Sometimes this feedback will be positive and very rewarding. For example, you might say, "I wanted to tell the computer how to draw a picture of a house with smoke coming out of a chimney. I wanted more smoke to come out of the chimney as I move from card to card in the stack. My program works just exactly like I wanted it to. Now I think I will add a bird to the stack, and have it fly away as the smoke gets thicker."

At other times the feedback will be mixed or negative. You might say, "I wanted to have the computer draw a plane flying in a cloudy sky. Right now my plane looks more like a car and my clouds seem to be sitting on the ground. I'm making progress, but it's clear I've still got some problems with my stack."

Activities

1. Create a stack of four cards, with each card containing a distinct graphic. Then reorder the stack so that your original fourth card

becomes the first card, the original third card becomes the second card, and so on. Figure out at least two quite different ways to accomplish this task. Which is easier or more efficient?

Note that this is a rather difficult problem for many people. This type of problem-solving task is much different from what one usually finds in school books. There, a particular type of problem is solved in a chapter or section of the book, and then you are asked to do a number of additional "exercises" just like the one done in the book. For Activity 1, if you need a hint, think about enlarging the stack to five cards in the process of completing the assigned task.

2. Do some journal writing about trial-and-error learning and problem solving. Include personal examples from your past as well as examples encountered as you studied this chapter.

3. Pick a discipline that you know well. Think about the types of problems that people solve in this discipline. Make a list of some of the most important primitives in this discipline.

4. *Constructivism* is a part of learning theory that says that new knowledge is constructed upon (built upon, based on) the knowledge you already have. The focus is on the learner rather than on the teacher and the teaching process. Different learners learn different things (construct different sets of knowledge for themselves) when presented with the same material. Think about the material in this chapter from a constructivist point of view. Write about the ways in which the new material fits in with the knowledge you already had before you began the chapter.

5. Think about the approach you used in attempting to learn the material in this chapter. Write down and analyze the approach you used. Did it seem effective to you? Did you build on your previous knowledge of computers? What might you do differently if you were starting all over again on this chapter? What might the book's authors have done differently to better help you learn the material in this chapter? (This is a metacognitive activity.)

6. How can you tell if you have learned "the right things" from studying this chapter? Create a test for yourself that covers what you feel are the right things to learn from this chapter. (This is a metacognitive activity.)

7. Analyze the test you created in Activity 6. Does it focus on small details, such as "Which menu contains the **New Card** option?" Or does it focus on larger concepts, such as "Discuss similarities and differences between *HyperCard* cards and index cards"? Is your focus on the trees or on the forest?

Chapter 5. Paint Tools

When you are thoroughly comfortable with the ideas from the previous chapter, which included creating a new stack with several cards, naming the cards, editing the stack, and moving from card to card, you are ready to move on. In this chapter you will learn more about using the Paint tools.

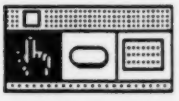
The Tool window contains 15 different paint tools. These provide a great deal of capability and flexibility in creating and editing drawings. It is not necessary to have a detailed knowledge of the material in this chapter before proceeding to the subsequent chapters. However, many people thoroughly enjoy using the Paint tools and find them to be an exciting part of *HyperCard*.

The Paint tools can be used to help solve a wide range of drawing problems. Two key ideas in solving drawing problems are that you can provide your own feedback and it is easy to use the computer to make changes to a drawing. Skill in using the Paint tools comes with practice.

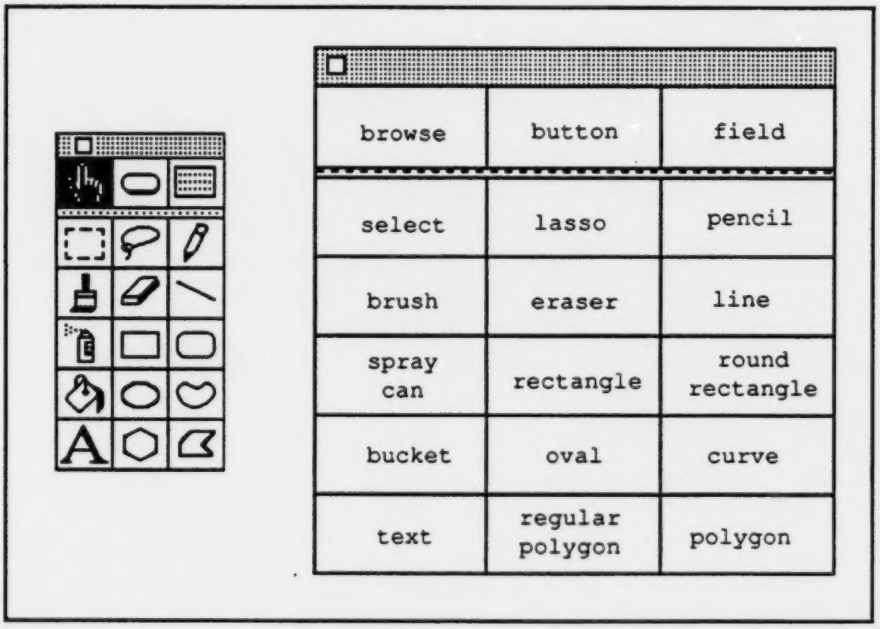
The Tool Window

In the previous chapter you developed skill at creating a new stack containing several cards. You learned how to "tear off" the Tool window and place it at a convenient location on your computer screen. The Tool window can be moved to a different location by placing the pointer on the top bar on the Tool window and holding the mouse button down as you drag the Tool window to a new location.

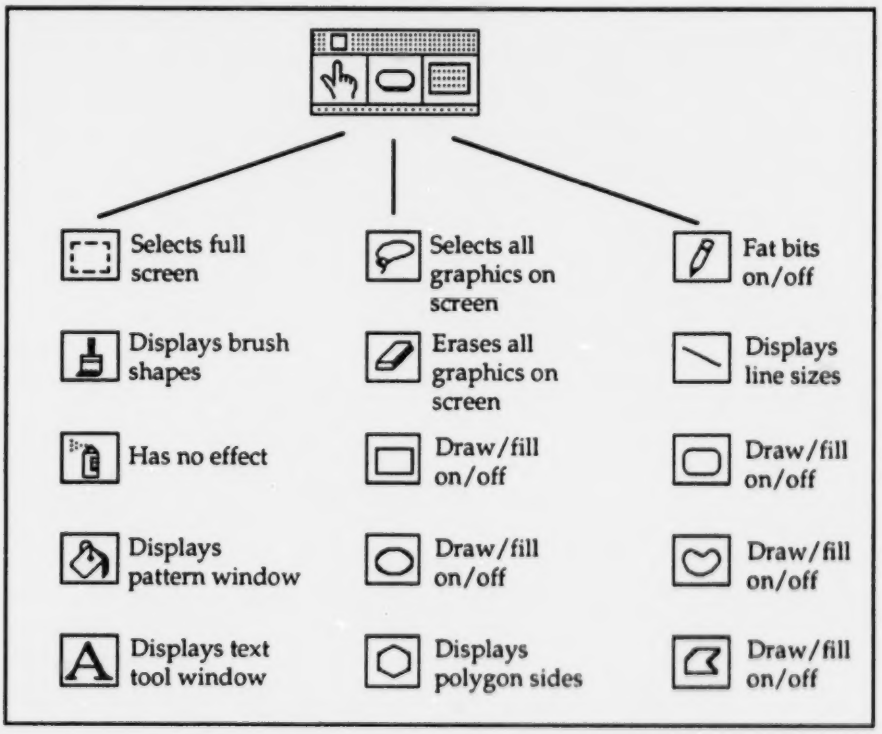
You have already learned to use the **Browse** tool located at the top left corner of the Tool window. In subsequent chapters you will learn to use the **Button** tool and the **Field** tool, the other two tools in the top row of the Tool window.



The following diagram shows the name and location of each of the 15 paint tools that are below the dotted line in the Tool window. These tools can be used create and edit graphics in *HyperCard*. On the left side of the diagram is the Tool window. On the right side, the name of each tool is given in the same location as its corresponding icon on the left.



Many of the Paint tools are actually multiple tools. Clicking once on a tool selects the tool. Double clicking on a tool provides you with further capabilities. The following diagram describes what happens when you double click on a tool.



Experiment with the various tools, using both the single-click and the double-click options. Be sure you have a card on screen that you don't mind modifying or destroying. Some of the details of the features of each of the 15 Paint tools are given in the remainder of this chapter.

Graphical User Interface (GUI)

One of the most important features in the design of the Macintosh computer is the Graphical User Interface (GUI). The Macintosh uses a range of graphical images, such as icons, menus, and windows, to help the user operate the computer system. For example, often there are icons that indicate choices of available actions. The human mind is quite good at recognizing and processing picture-type information. A well-designed GUI greatly reduces the need for memorization of sequences of keystrokes and is of particular help to beginners or to users who operate computers infrequently.

As you study the material in this chapter, think about the GUI features you are using. Do the menus make sense to you? Do the icons make sense to you? Are they conveniently located and easy to use? Do they help you accomplish the tasks you want to accomplish, and do they reduce the "load" on your memory? Do they help you to learn?

In subsequent chapters you will create stacks designed to communicate with yourself and others. *HyperCard* provides a powerful set of graphic tools. You will want to make good use of the graphics in your stacks to ensure that your *HyperCard* stacks communicate effectively.

Using the Paint Tools

Each of the following sections provides you with an opportunity to explore the use of one of the tools in the Tool window. To gain maximum understanding of the Paint tools, you should be sitting in front of a computer as you work through these sections.

The Select Tool

A single click on the **Select** tool in the Tool window changes the pointer into a dotted-line cross-hair pointer. This pointer is used in a click-and-drag movement to form a rectangle around an object on the screen. Everything within the rectangle is "captured." When the mouse button is released, a dotted marquee appears and moves around the rectangular area.

If the Delete key is pressed, everything that has been captured is erased. If the pointer is moved inside the rectangular captured region, it becomes an arrow. A click-and-drag movement can then be used to move the captured region. In addition, the **Copy Picture** and **Paste Picture** options from the **Edit** menu can be used on the captured region. This works even when the captured region is some clip art from a different stack that you want to paste onto a card in your stack.

A double click on the **Select** tool in the Tool window captures all of the graphics painted on the entire card. Pressing the Delete key then erases all these graphics. Note that the card is still there—it has just become a blank card. The **Undo** choice in the **Edit** menu can be used to restore the graphics.

The Lasso Tool

A single click on the **Lasso** tool in the Tool window changes the pointer into a dotted lasso. This lasso pointer is used to draw around the graphics to be selected. When the mouse button is released, a straight line is drawn from the pointer to the beginning of the line. Then the lasso collapses against the object.

The outline of the captured area appears to "shimmer" as the dotted line moves around it. If the Delete key is pressed, everything that has been captured is erased. If the pointer is moved inside the captured region, it becomes a arrow. A click-and-drag movement can then be used to move the captured graphic.

A double click on the **Lasso** tool in the Tool window captures all of the graphics painted on the entire card. Pressing the Delete key then erases all these graphics. Note that the card is still there—it has just become a blank card. The **Undo** choice in the **Edit** menu can be used to restore the graphic. If the pointer is moved to an object in the captured region, it turns into an arrow. A click-and-drag movement can then be used to move all the graphics.

The Pencil Tool

A single click on the **Pencil** tool in the Tool window turns the pointer into a pencil. This pointer is used in a click-and-drag movement to draw on the card.

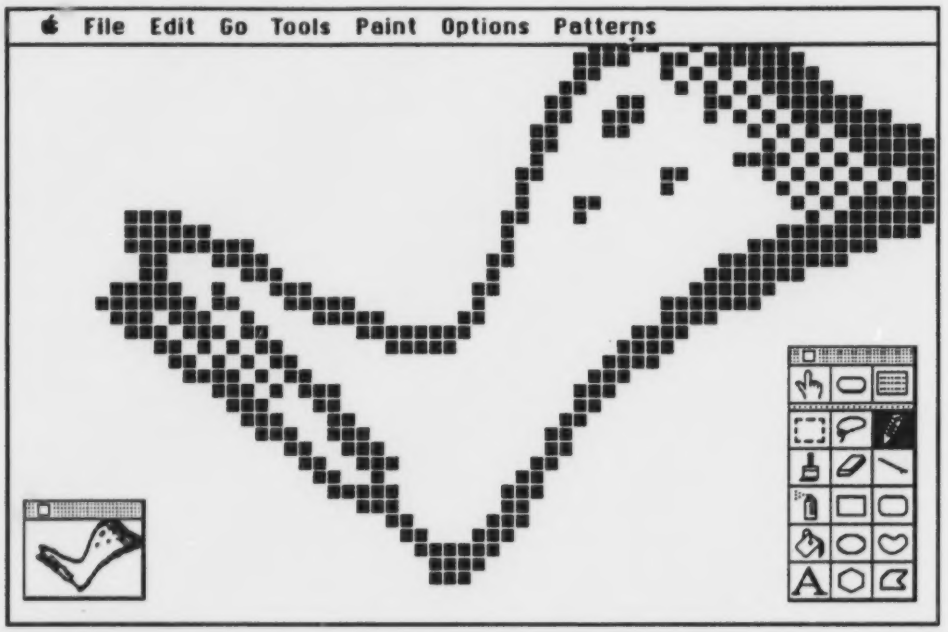
The drawings produced by a number of the tools are constrained when the Shift key is held down during the click-and-drag movement. For example, if the Shift key is held down while using the **Pencil** tool, only horizontal and vertical lines can be drawn.

A double click on the **Pencil** tool in the Tool window magnifies the card graphic into FatBits. That is, the graphic is expanded so that it can be edited at the individual picture-element level. At that level, clicking on a square turns it from on to off, or from off to on. A click-and-drag movement can be used to draw. Select the **Browse** tool or double click again on the **Pencil** tool to return to a normal view of the card.

In FatBits, only a small part of the entire card is visible at any one time. A small window in the display shows that portion of the entire card. If you hold down the Option key, the pencil pointer will become a hand. This can be used in a click-and-drag movement to move the view so that it is easy to view different parts of the card graphic.

FatBits can be used to edit any paint graphic. Suppose you have copied the following clip art onto one of your cards and you want the dinosaur to have several more dark spots on its back.

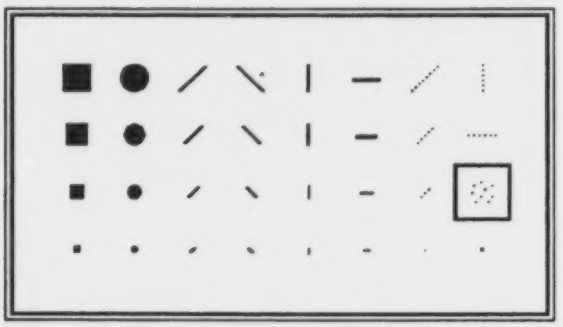
The following graphic is the FatBit representation of the dinosaur. It shows the right foot of the dinosaur. You can hold the Option key down and move the mouse to move the view to the spots on the dinosaur's back.



The Brush Tool

A single click on the **Brush** tool in the Tool window turns the pointer into a paint brush. This is used in a click-and-drag movement to paint on the card. If the Shift key is held down while using the **Brush** tool, the brush is constrained to horizontal or vertical movement.

A double click on the **Brush** tool in the Tool window brings up a menu of brush shapes. Click on the particular brush shape you want to use. The following diagram shows a square around the currently selected brush shape.



The Eraser Tool

A single click on the **Eraser** tool in the Tool window turns the pointer into an eraser. This is used in a click-and-drag movement to erase graphics on a card. If the Shift key is held down while using the **Eraser** tool, the

eraser is constrained to a horizontal or a vertical movement. Note that the **Eraser** tool can be used in FatBits.

A double click on the **Eraser** tool in the Tool window erases all of the graphics on a card. Note that the card is still there—it has just become a blank card. The **Undo** choice in the **Edit** menu can be used to restore the graphic.

The Line Tool

A single click on the **Line** tool in the Tool window turns the pointer into cross hairs. This is used in a click-and-drag movement to draw straight lines. A line is drawn from the point where you click to the point where you release the mouse button.

If you hold down the Shift key while drawing a line, the line will be constrained in 15 degree increments. That is, if you draw a line that is nearly horizontal, it will be exactly horizontal. If you draw a line that is nearly at a 30 degree angle, it will be exactly at a 30 degree angle.

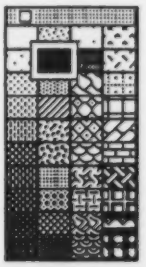
A double click on the **Line** tool in the Tool window brings up a menu of available line widths. Click on the line width that you want to use to draw lines. In the following diagram, a small rectangle surrounds the currently selected line width.

The Spray Can Tool

A single click on the **Spray Can** tool in the Tool window turns the pointer into a spray can. This is used in a click-and-drag movement to spray paint onto the card. If the Shift key is held down while using the **Spray Can** tool, spraying is constrained in a horizontal or a vertical direction.

A double click on the **Spray Can** tool in the Tool window has no effect.

The **Patterns** window contains a large number of patterns that can be selected for use with the **Spray Can** tool. This window is a tear-off menu, like the Tool window.



Experiment with selecting different patterns for the **Spray Can** tool. Also experiment with moving the mouse quickly and slowly in the click-and-drag spraying movement to produce different graphic effects.

The Rectangle Tool

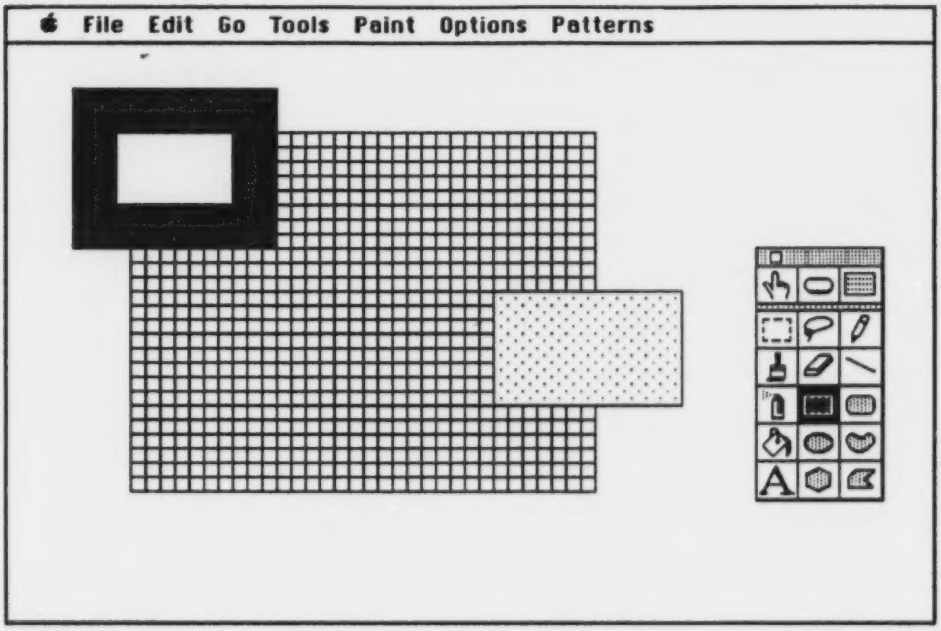
A single click on the **Rectangle** tool in the Tool window turns the pointer into cross hairs. This can be positioned at any point on the card and then used in a click-and-drag movement to draw a rectangle. If the Shift key is held down while drawing a rectangle, a square will be produced.

There are several important options in drawing rectangles. Recall how to change the line width by double clicking on the **Line** tool. Alternatively, the **Option** menu has a **Line Size** choice. The **Rectangle** tool draws using the line width that has been selected.

A double click on the **Rectangle** tool results in "filled" rectangles being drawn. The rectangle is filled with the pattern that is selected in the **Patterns** pallet. Thus, you can draw rectangles filled with quite interesting patterns. The choice of patterns is discussed in the **Spray Can** section above.

The Tool window contains six tools that can be used in a Draw Filled mode. They are the six tools in the bottom right corner of the Tool window, starting with the **Rectangle** tool. When these six tools are shaded gray, the Draw Filled mode has been selected. It can be selected by choosing Draw Filled from the **Options** menu, or by appropriately double clicking on one of the six tools (unless the double click has a specific other meaning, as it does for the **Regular Polygon** tool).

The following graphic was created by using the **Rectangle** tool with the Draw Filled option. Four different rectangles were drawn, each with a different pattern.



The Round Rectangle Tool

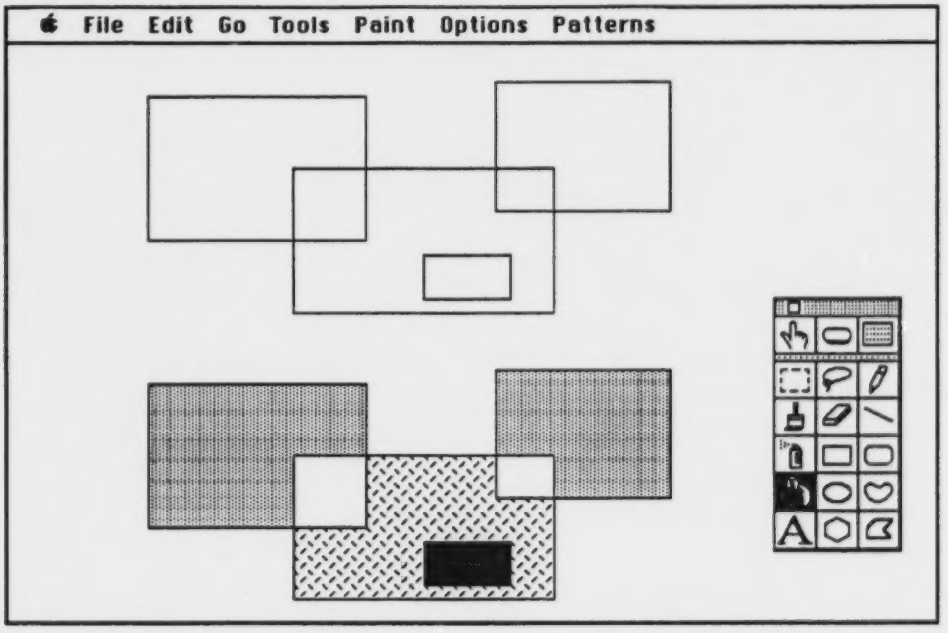
The **Round Rectangle** tool in the Tool window works exactly like the **Rectangle** tool. It produces rectangles with rounded corners. Remember that this is one of the tools that has a Draw Filled option. In addition, you can draw using different line widths.

The Bucket Tool

A single click on the **Bucket** tool in the Tool window turns the pointer into a bucket. Think of it as a bucket of "pattern" that can be poured into a closed container. The pattern that will be poured is the pattern that has been selected on the **Patterns** window. The **Patterns** window can be selected either by double clicking on the **Bucket** tool or by using the **Patterns** menu at the top of the screen.

A pattern is poured out of the bucket by positioning the bucket over a closed container (a graphic that is completely closed) and clicking the mouse. If the container is not closed, the pattern will fill a larger enclosed region—which may mean filling the entire card. Remember that the **Edit** menu contains an **Undo** choice!

The top part of the following graphic was created by drawing four rectangles. Notice that the graphic contains six distinct closed containers. The bottom part of the graphic depicts the result of pouring different patterns into some of the containers.



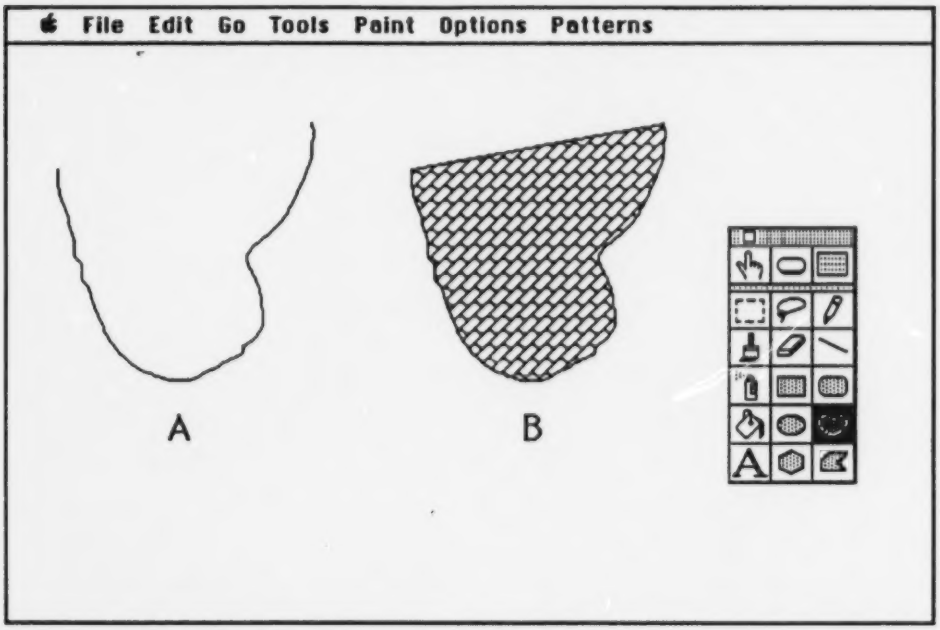
The Oval Tool

The **Oval** tool in the Tool window works exactly like the **Rectangle** tool. It is used to produce oval shapes. If the Shift key is held down during the click-and-drag movement, the oval is restricted to being a circle. Remember that this is one of the tools that has a Draw Filled option and that you can draw using different line widths.

The Curve Tool

A single click on the **Curve** tool in the Tool window turns the pointer into cross hairs. This pointer can then be used to draw in exactly the same manner used to draw with the pencil. However, it will draw with the line width that has been most recently selected.

A double click on the **Curve** tool in the Tool window turns on the Draw Filled option. Whatever curve you then draw will automatically be completed into a closed container by the computer, and it will be filled with the most recently selected pattern. The computer will produce a closed area by assuming that a line has been drawn from your starting point to your ending point. Thus, when figure A on the left in the following graphic was drawn in a Draw Filled mode using the "brick" pattern, it produced figure B on the right.

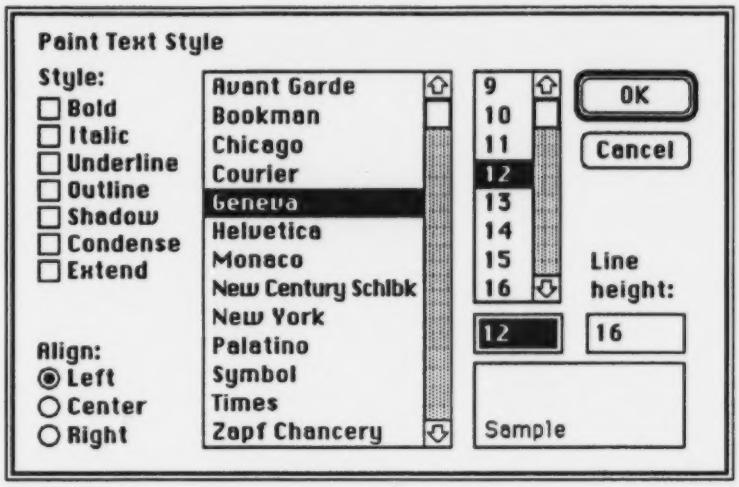


The Text Tool

A single click on the **Text** tool in the Tool window turns the pointer into an I-beam. Position this pointer where you want to place text on the card, click, and begin keying in the text. The text will be "painted" onto the card. This means that it can be erased using the **Eraser** tool. It also means that, unlike word-processed text, it *cannot* be edited.

For example, if you position the pointer in the middle of some text on the card and use the Delete key, nothing happens. The exception to this is that when you are initially keying in text, the Delete key can be used to erase characters that have just been keyed in. It takes a little practice to get used to the idea that text is edited in the same way that other parts of a graphic are edited. For example, you can select text you have placed on a card by using the **Select** tool or the **Lasso** tool. Then it can be moved or deleted, just like any other piece of a graphic can be moved or deleted.

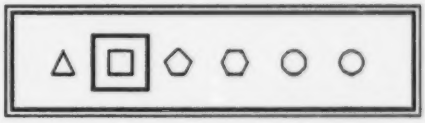
A double click on the **Text** tool in the Tool window brings up the following Font dialog box. With it, you can select style, font, font size, and so on.



The Regular Polygon Tool

A single click on the **Regular Polygon** tool in the Tool window turns the pointer into a polygon. This is used in a click-and-drag movement to draw a regular polygon. Notice that you can rotate the polygon as part of the click-and-drag movement. As with all six of the tools that have a Draw Filled option, the **Regular Polygon** tool allows drawing with different line widths. If the Shift key is held down while drawing a regular polygon, the positions of edges are constrained at 15-degree angles.

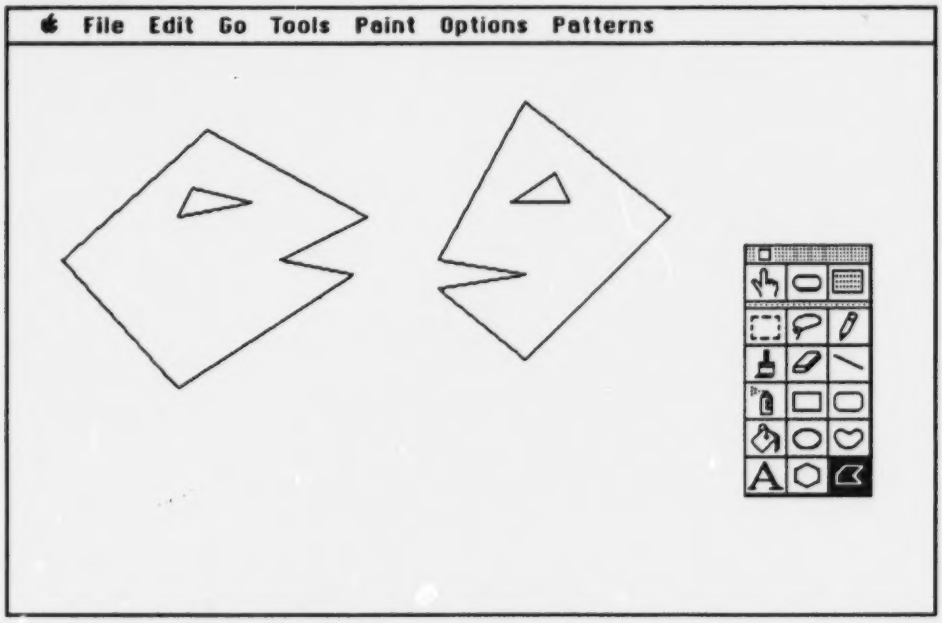
A double click on the **Regular Polygon** tool in the Tool window brings up a menu of polygon shapes. (The same selection is obtained through choosing **Polygon Sides** in the **Options** menu.) There are six different regular polygonal shapes available. In the following graphic, the square has been selected.



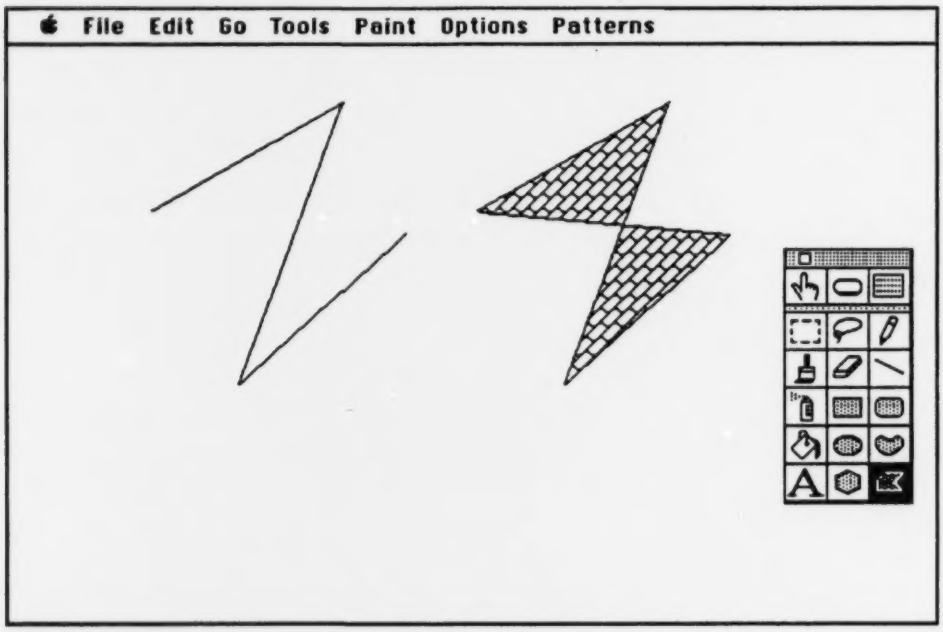
The Polygon Tool

A single click on the **Polygon** tool in the Tool window turns the pointer into cross hairs. This pointer is used in a sequence of click-and-drag movements to draw a sequence of line segments. Begin by clicking and releasing the mouse button on the card at the position where you want the drawing to begin. As you drag the mouse, a line segment is drawn. Click the mouse to end this line segment and to begin another one that starts where the previous one ended. Continue this process until the

drawing is complete, and then double click. You can make drawings such as the following.



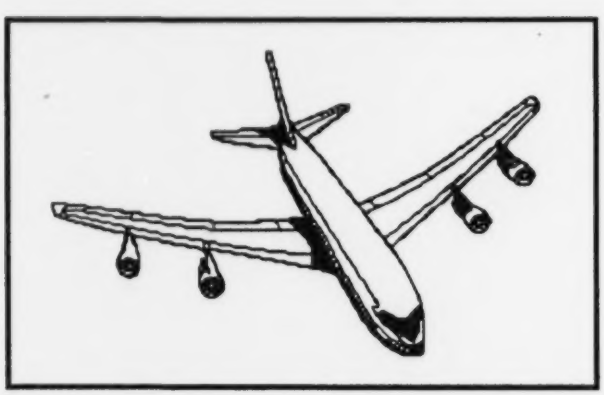
A double click on the **Polygon** tool in the Tool window turns on the Draw Filled option. Whatever polygonal line shape you then draw will be automatically completed into a closed container by the computer, and it will be filled with the most recently selected pattern. The computer will produce a closed container by assuming that a line has been drawn from your starting point to your ending point. In the following graphic, use of Draw Filled on the figure on the left produced the figure on the right.



Clip Art

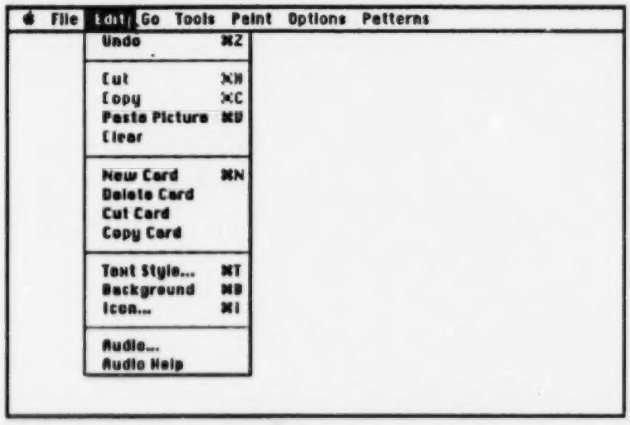
The Home Card that accompanies *HyperCard* typically displays icons representing various stacks that accompany that version of *HyperCard*. Usually there are one or more stacks of clip art, art objects, scanned images, or other forms of graphics that can be copied and used in your own stacks.

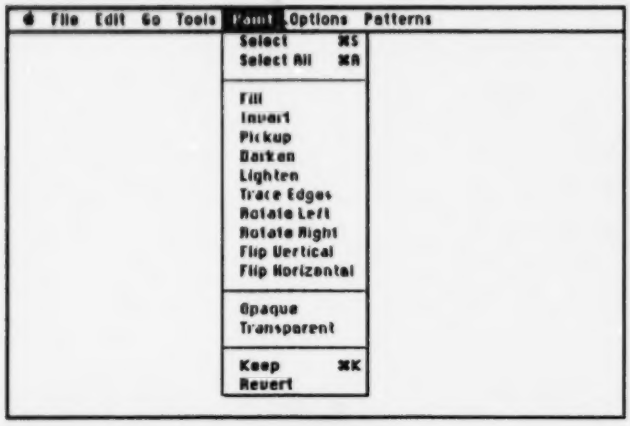
Select **Home** from the **Go** menu. Explore one of the art stacks. While in the stack, use the Select tool or the Lasso tool to select a graphic. Use **Copy Picture** in the **Edit** menu. Then return to your own stack and use **Paste Picture** from the **Edit** menu. Remember that the **Recent** choice in the **Go** menu is quite useful in finding your way back to the card from where you started in your stack. Here is an example of clip art that may be in your version of *HyperCard*.

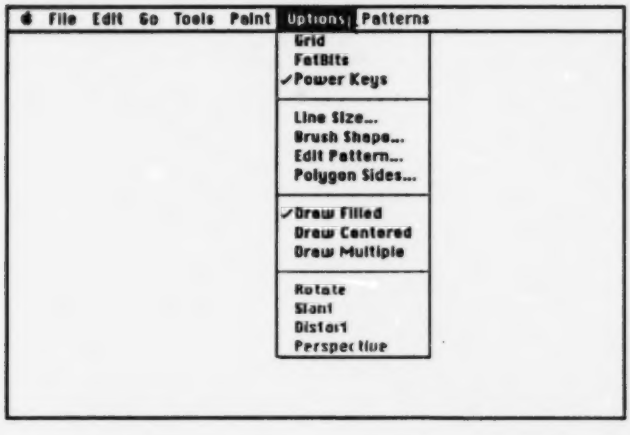


Menus

The Paint tools are a very powerful feature of *HyperCard*. We have not covered all of the features. Examine the **Edit**, **Paint**, and **Options** menus. Each contains choices useful in doing drawings using the Paint tools. The choices that are available for your use at a particular time appear in **boldface** in the menus.







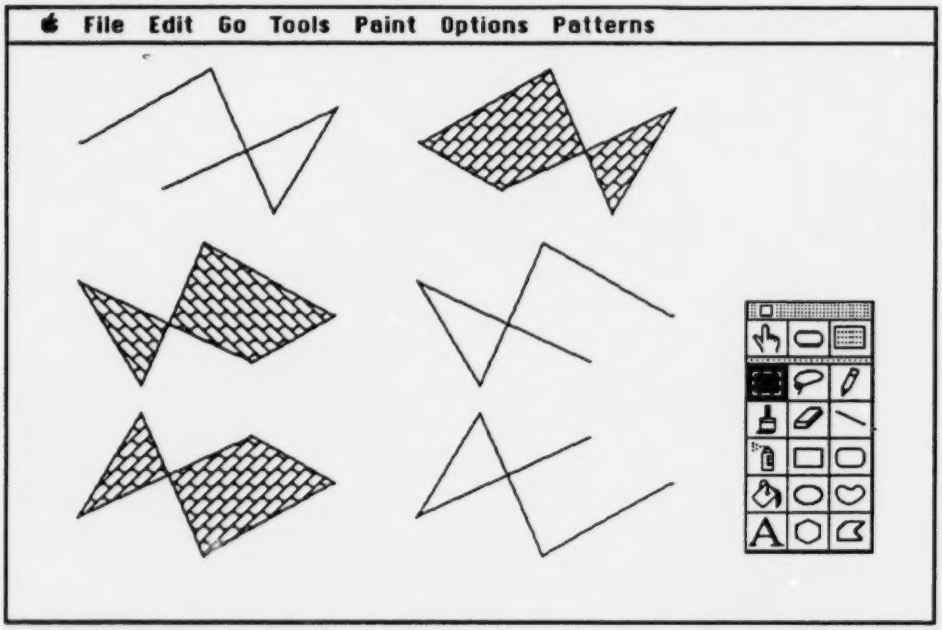
For example, to use **Cut Picture**, a piece of a graphic needs to have been selected by using the **Select** tool. (Note that the choices available may vary a little with the particular version of *HyperCard* you are using.)

Notice that the **Edit** menu contains **Copy** and **Paste** options. A graphic can be selected, pasted, and moved by dragging (it remains selected after pasting) to a desired location. Holding down the Shift key while dragging constrains the movement to horizontal or vertical directions.

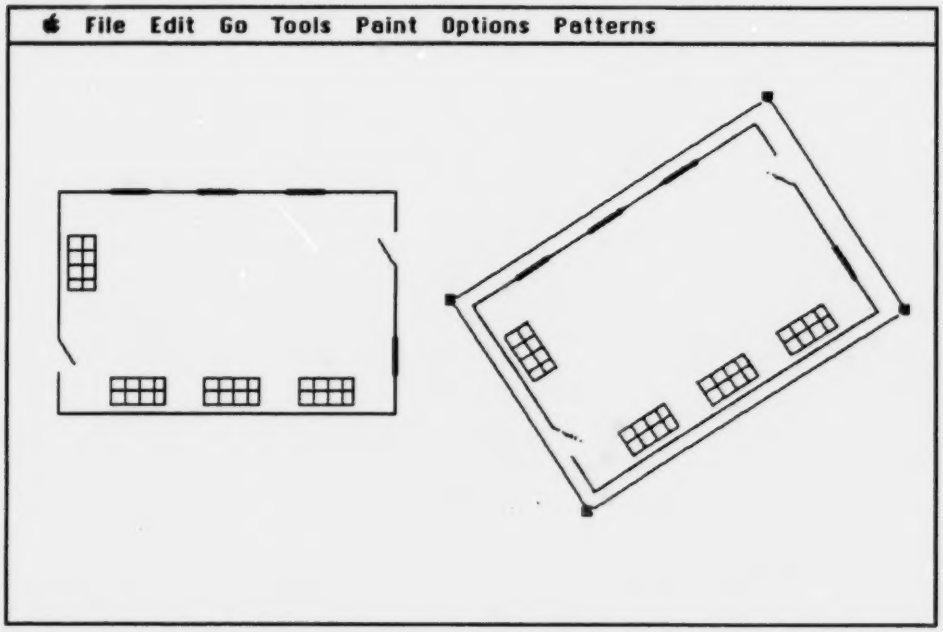
The **Keep** and **Revert** options in the **Paint** menu are useful in undoing mistakes or experiments while developing a graphic. Suppose you have created a graphic on a card. When you move on to another card, the card you have created is saved. (Alternatively, **Keep** causes a copy of the graphic to be saved.) Suppose you begin editing a graphic that has been saved. After working on it for a period of time, you decide that you would rather have the original (saved) graphic. **Revert** will replace the current graphic with the saved graphic.

Take a look at the **Flip Horizontal** choice in the **Paint** menu. When a graphic has been selected (for example, by using the **Select** tool or the **Lasso** tool), **Flip Horizontal** does a horizontal flip of the graphic. The "flipping" occurs in the space where the graphic is located.

The steps used in making the following graphic were first to create the top third of the graphic, then make a copy of it, and then paste the copy in the location of the middle third of the graphic. Notice that the middle third of the graphic is a horizontal flip of the top third of the graphic, and the bottom third is a horizontal flip of the middle third. You will want to do extensive experimentation with the wide range of choices in the **Paint** menu.



Examine the **Rotate** choice in the **Options** menu. Select a graphic by using the **Select** tool and then select **Rotate** from the **Options** menu. The graphic can be rotated by using the mouse to grab one of the four handles. The following figure was produced by making a copy of the figure on the left, pasting it in, moving it to the right, and then rotating it.



Problem Solving

Think about the problem situation of learning to use the Paint tools in *HyperCard*. This learning and problem situation has two major components. First, there is the task of learning the rudimentary mechanics and uses of each tool. Second, there is the task of using the tools for designing and developing *HyperCard* stacks that communicate effectively with yourself and others.

You are aided in the first task by the Graphical User Interface that the *HyperCard* designers have developed for this purpose. Moreover, it is not necessary to learn about each Paint tool before beginning to use some of the tools. You can learn by doing; you can use trial and error; you can do discovery-based learning. You can easily detect your own errors ("That doesn't look right. I intended to draw a rectangle and instead I drew a triangle.") and correct them on the spot ("Thank goodness for **Undo** and **Revert**."). When the need arises, you can learn more tools and more about the tools. It is not necessary to master all of the tools before proceeding further in the book or attempting to solve problems of interest to you.

The second task—designing artwork that communicates—is more daunting. It may be that you have had a great deal of artistic design training. You may be a skilled artist. You may be able to glance at a graphic and "see" its good and bad features. If you have these skills, you know they were obtained through a combination of natural talent and a great deal of hard work. You will probably find that it is relatively easy to work in this new medium—the medium of the Paint tools.

If you are not a trained artist, it is unreasonable to expect that access to the Paint tools will somehow magically make you a skilled artist. Being a skilled artist is an example of domain specificity in problem solving. The Paint tools are an example of domain independence. The Paint tools are useful over a wide range of problem-solving domains. To use them skillfully in any particular domain takes both learning and experience.

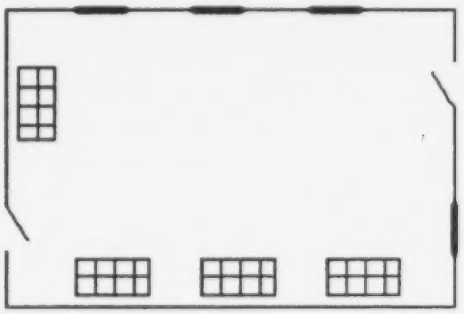
Activities

1. Think about a particular Paint tool and a graphic that can be created using just that one tool. Design the graphic in your "mind's eye" and/or do a rough sketch of it on a piece of paper. Then create the graphic using the Paint tool. Repeat this activity for several different Paint tools. (Creating something in your mind's eye is a mental modeling process—a key idea in problem solving. Modeling is discussed more in Chapter 11.)

2. Think about a particular graphic that you feel you can probably create using your current knowledge of a combination of several of the Paint tools. Design the graphic in your "mind's eye" and/or do a rough

sketch of it on a piece of paper. Then create the graphic using the Paint tools. Repeat this activity for a different graphic.

3. The following graphic is the beginning of a design for the layout of a room. Use the Paint tools to do a floor plan for a room layout or for a house. You may find it helpful to take advantage of the **Grid** choice in the **Options** menu. This places an invisible grid, with eight-pixel spacing, on the card and automatically restricts the pointer to the grid points. (A pixel is a "picture element"—a single dot on the screen display.)



4. It is possible to do animation using *HyperCard*. Animation is discussed in Chapter 11. The activity given here illustrates a rudimentary type of animation.

   Create a new stack of four or five cards. Each card is to contain the same graphic, but the graphics should be in slightly different positions. The idea is to simulate motion by moving quickly from card to card. Use the ⌘-3 (Command-3) keystroke to move from card to card.

   In problem solving, one gains a great deal of power by having the right tools and appropriate knowledge in using the tools. When you are attempting to solve a problem using *HyperCard* and you don't seem to be able to figure out how to do it, this may be due to *HyperCard* being an inappropriate tool for the problem and/or your not yet having enough knowledge of *HyperCard*.

5. Write about the general idea of a Graphical User Interface. Give some examples of where you have encountered a GUI other than in your use of computers (for example, traffic signs). Discuss advantages and disadvantages of a GUI as an aid to learning, doing, and relearning.

6. Consider the ideas of domain specificity and domain independence in problem solving, as discussed near the end of this chapter. Discuss particular examples of this from your own life. The examples need not be restricted to the ideas about artistry discussed in this chapter.

7. Discuss advantages and disadvantages of creating graphics using the Paint tools in *HyperCard* versus creating the same graphics using noncomputer tools such as pencil, paper, straight edge, compass, and protractor. In your discussion, consider the transfer of learning from one of these types of situations to the other; also, discuss ease of correcting errors.

8. One of the most important ideas in problem solving involves building on the previous work that you and others have done. Often this requires modifying the previous work so that it fits the current situation. Discuss the Paint tools in *HyperCard* from this point of view. Suggest additional features in the graphics tools that might be especially useful both in creating original graphics and in building upon previously created graphics.

Chapter 6. Buttons

A button is one of the five types of objects in *HyperCard*. In the stacks you examined in Chapter 3, you encountered a variety of buttons. You used a mouse to click on these buttons to take various actions, such as moving to the next card in a stack or returning to the Home Card.

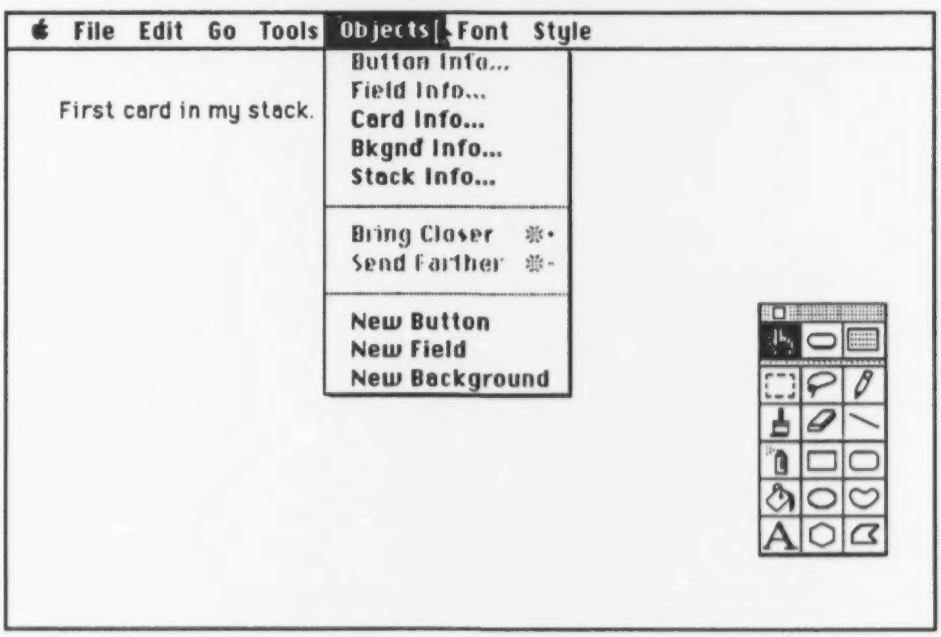
Buttons are used to facilitate interaction between a *HyperCard* stack and the stack user. The stack user can make choices and click on buttons to communicate these choices to the *HyperCard* system. Often these choices result in moving through the cards of a stack in a nonsequential manner.

In this chapter you will learn to create buttons and use them in building a stack. You will be introduced to some of the design considerations that will help you effectively use buttons in stacks. Good design is essential to effective communication.

Creating a Button

Buttons are commonly used to link cards so that you can easily move from one card to another without using the **Go** menu or keypress combinations.

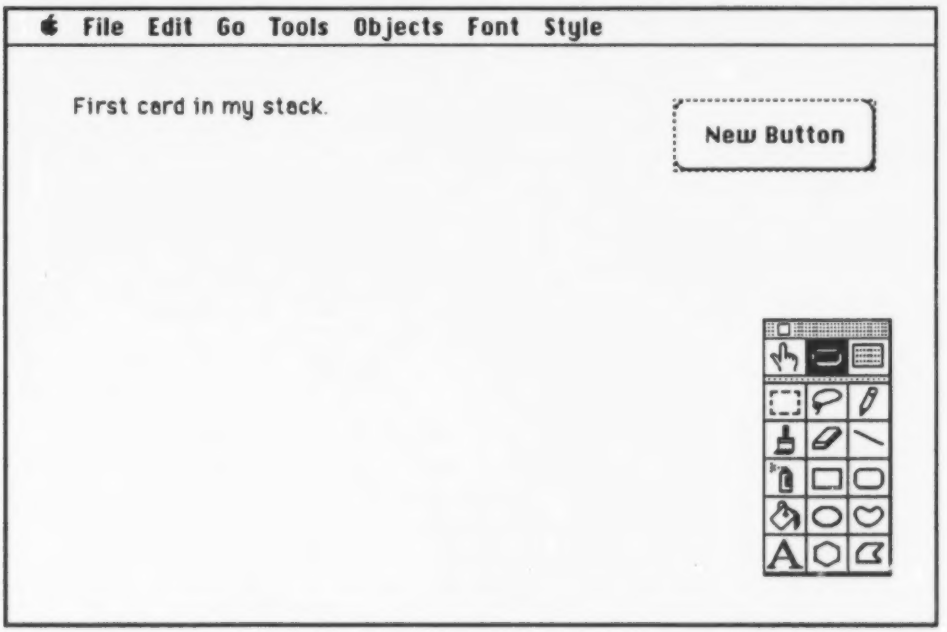
To get started exploring buttons, create a stack of at least three cards or open a stack you have previously created that has three or more cards. Go to your first card and choose **New Button** from the **Objects** menu.



When you choose **New Button**, a button will be placed on your card. The actual shape of the object produced is rectangular. (You can see the dashed, rectangular-shaped outline of the button displayed on the card at the time the button is created and/or when the button is selected by using the **Button** tool.)

Technical Note: A button has a "hot spot" that is rectangular in shape. The actual appearance of the button may be different than the rectangular-shaped hot spot.

You can position a button at any desired location on the card by clicking and dragging near the middle of the button. You can change the size of a button by using the mouse to "grab" a corner of the rectangle and then dragging it to the desired size.

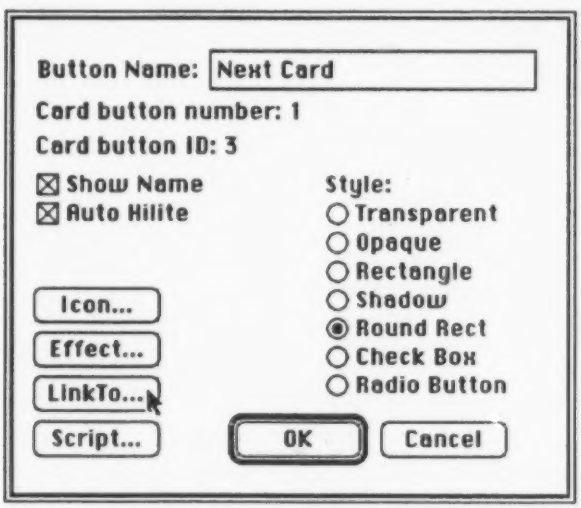


Next, choose **Button Info** (or you can double click on the new button). This gives you a dialog box of choices for the new button. For *HyperCard* 2.0, the dialog box may appear as follows.

You can give the button a name, such as Next Card. Notice the **Show Name** choice on the left side of this dialog box. If **Show Name** is selected, the name of the button will appear on the button itself. Thus, you want to give the button a name that will communicate effectively with people who use your stack.

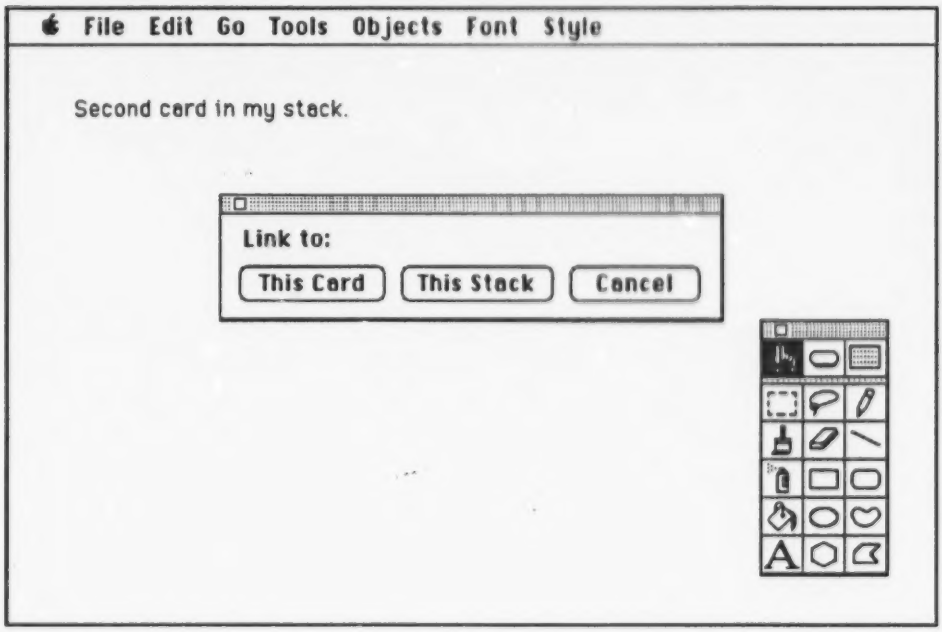
You can indicate the style of the button, such as Round Rectangle. You will want to experiment with these different styles. For example, observe that a Transparent button with Show Name not checked is completely invisible. Notice that an Opaque button with Show Name not checked can cover up a spot on a card—even another button.

Notice the **Auto Hilight** choice. This is used to specify that when the user clicks on the button, the button is highlighted and then returns to normal when the mouse button is released.



Notice that there are a number of additional options in the bottom left corner of the menu. The **LinkTo** choice is used to "link" the button to another card. That is, it is used to specify which card to move to when the user clicks on the button. Select the **LinkTo** choice and you will get a dialog box like the following.

Next, use the **Go** menu or the appropriate keystroke commands to move to the card you want to link to. Note that the LinkTo dialog box stays on the screen. In this case, you want the button to link to the second card in your stack. When the second card of your stack is visible, click on **This Card** in the dialog box.



That's it! You have now created an active button that links your first card to your second card. Select the **Browse** tool and click on your button. This will move you from your first card to your second card. Note that the **Browse** tool *must* be selected in order for the button to be active.

Practice these ideas by creating a button that moves the user from your second card to your third card. Then add a second button to your first card that moves the user directly from your first card to your third card.

Eventually you will become very skilled at creating buttons and linking them to cards. The next section contains some shortcuts that help speed up the process of creating and linking buttons.

Copying, Pasting, and Deleting a Button

Notice the **Cut Button**, **Copy Button**, and **Paste Button** options in the **Edit** menu. These are all active when a button has been selected (for example, by clicking once on a button when the **Button** tool is selected). The **Cut Button** or the Delete key is used to delete a button that has been selected.

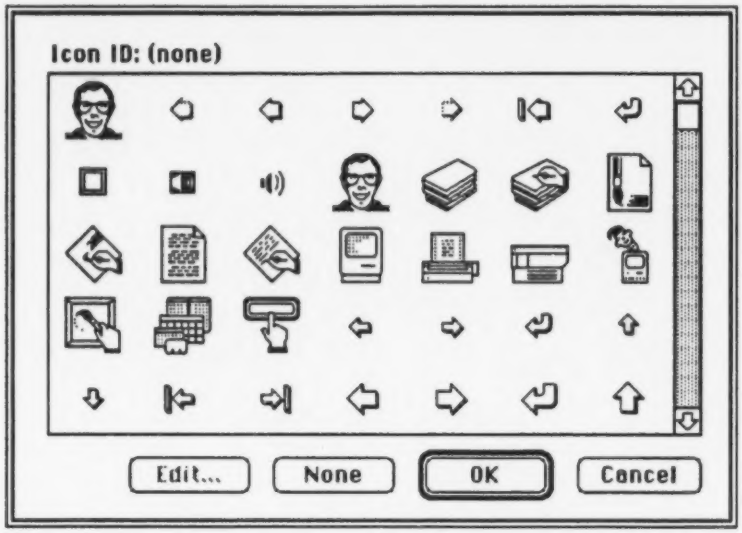
Suppose you have a stack and you want buttons that look exactly alike in the same location on each card. This can be done by using the **Copy Button** and **Paste Button** options in the **Edit** menu. Start on your first card, create and position the desired button, and link it to another card (for example, to the second card).

While the button is selected (remember to choose the **Button** tool and click once on the button you want to use), select **Copy Button** from the **Edit** menu. Then go to the second card in your stack and do a **Paste Button**. Link this button as desired. Then go to your third card and select **Paste Button** from the **Edit** menu. Link this button as desired, and continue pasting the button on subsequent cards.

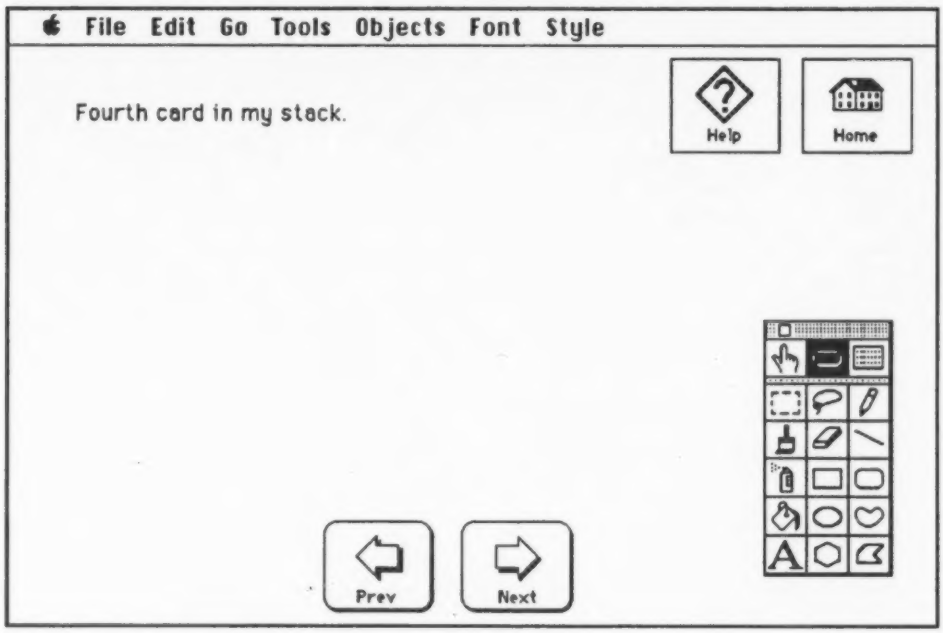
Using keystroke commands makes the cut-and-paste operation even faster. In fact, it is so easy to copy and paste buttons that you may forget to link the pasted buttons to the appropriate card! If you do, you will have buttons that link to the wrong place.

Button Icons

The *HyperCard* system contains a number of icons you can use as buttons. Select a button and choose **Icon...** in the **Button Info** dialog box. This will give you an Icon dialog box containing a large number of button choices.



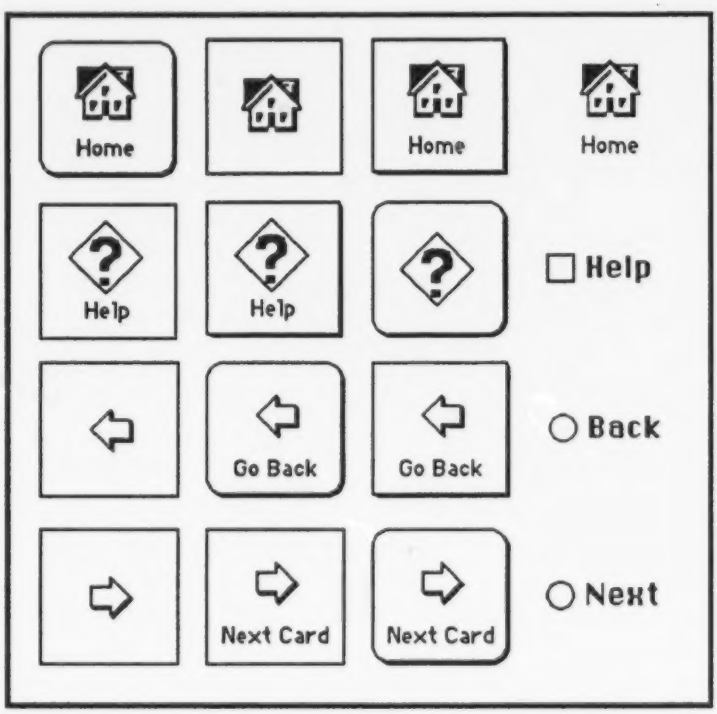
When you click on one of these button icons, the icon is placed on the selected button. The button itself is still a rectangle and can still be sized and moved, as discussed previously in this chapter. The following card contains four buttons, each with a different button icon.



Button Conventions

The choice of button style and button icon contributes significantly to the overall appearance of a card. There are a number of button conventions that are usually followed when using button icons. The intent

is to make it easier for people to use a stack. Consider the following buttons:

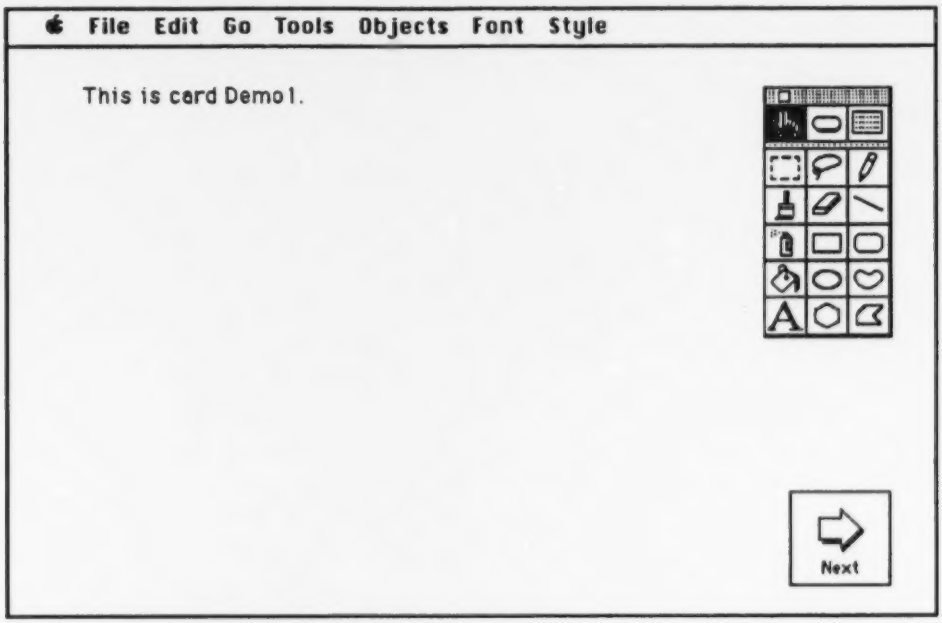


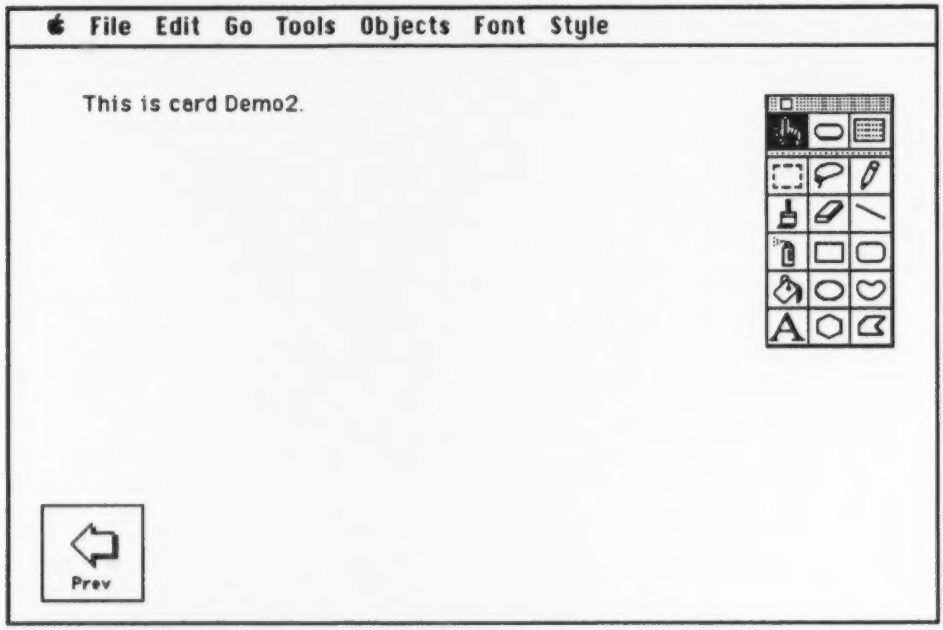
There are four types of icon buttons on the card shown in the previous graphic. Each has a commonly agreed upon meaning. It is common to use a house icon button to move to the Home Card. It is common to use a question mark icon button as a Help button. It is common to use the left-pointing arrow button at the bottom of the card to move to the previous card and the right-pointing arrow button to move to the next card.

Suppose that *every* person who creates *HyperCard* stacks agreed that a question mark icon button would always be a Help button. That is, a question mark icon would always mean "Click on me if you have a question about what you should be doing." After a few uses of such a button, the typical user would memorize its meaning. While the name Help could still be there as a reminder, the typical user would soon find that the icon by itself suffices to convey the message.

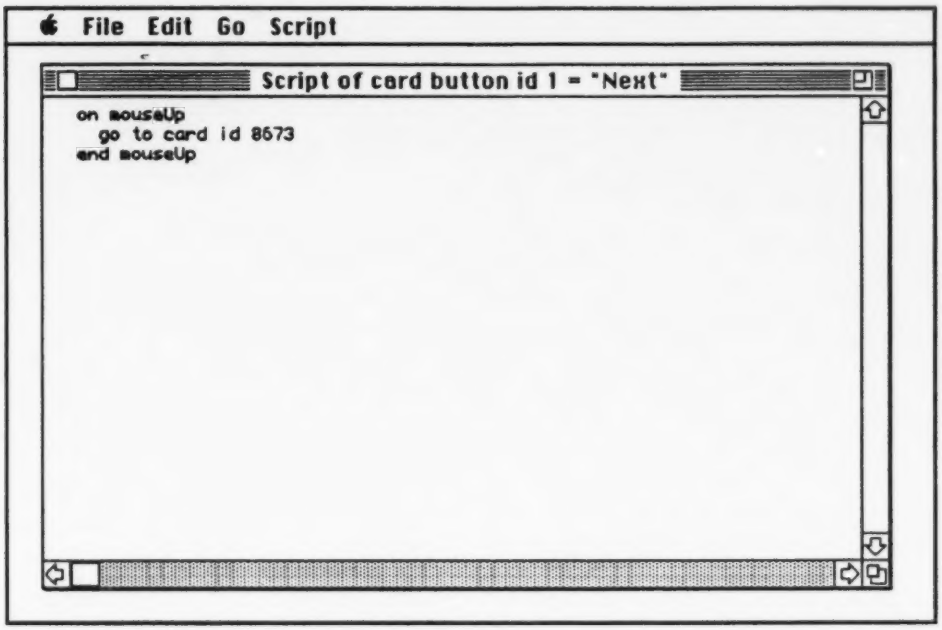
Scripting Buttons

This section assumes that you have a stack where the first card is named Demo1 and the second card is named Demo2. The first card contains a button named Next that is linked to the second card. The second card contains a button named Prev that is linked to the first card.





Select the button named Next on the card Demo1. (Remember, this can be done by selecting the **Button** tool and clicking on the button.) Then choose the **Script...** option on the **Button Info...** menu. You will then see something like the following. (The screen will look somewhat different if you are using a version of *HyperCard* earlier than 2.0.)



As shown in the title bar of this graphic, the title of the scrolling field is **Script of button id 1 = "Next."** This contains the button id and its name. There are three lines of computer code given. Notice the scroll bar on the right side of the field. If a large number of lines of code are entered into the field, it will scroll. These tell *HyperCard* what to do when the mouse pointer is on the button named Next. The line

```
on mouseUp
```

tells *HyperCard* to look for the release of a mouse button. The second line

```
go to card id 8673
```

tells *HyperCard* to go to the specified (next) card.

This script can be edited, much like one edits a word-processor document. Change the second line of the script so that it reads

```
go to card "Demo2"
```

Then click on the close box or press the Enter key (not the Return key). You can also choose **Close Script** from the **File** menu or use the keystroke ⌘-W to accomplish the same task. Click on Yes when asked if you want to save the changed script.

Congratulations! You have modified one line of a script! Check to make sure that the button works as expected. Then examine the script for the button Prev on the card Demo2. Change the second line of this script so that it reads

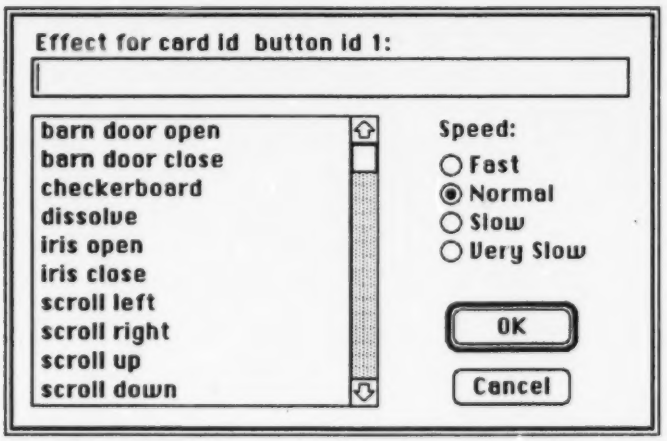
```
go to card "Demo1"
```

Visual Effects

Technical Note: HyperCard *versions numbered 2.0 and higher contain a Menu option that allows the specification of a number of different visual effects in moving from one card to another card. In earlier versions of* HyperCard, *the menu of visual effects is not available. However, the visual effects are easily added to button scripts.*

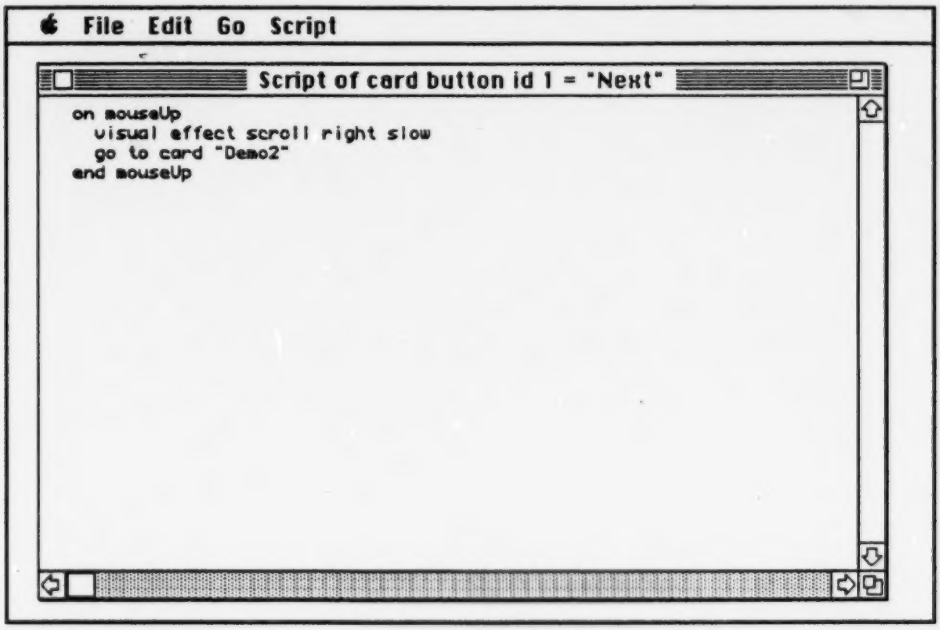
The first part of this section assumes you are using a version of *HyperCard* numbered 2.0 or higher. It discusses the addition of visual effects to a button by using a menu of visual effects. Later in this section button scripts that result from choosing from one of these visual effects will be shown. These visual effects can be written into button scripts. This is necessary to produce the visual effects in earlier versions of *HyperCard*.

The *HyperCard* system (2.0 and higher) makes it easy to add a visual effect when using a button to move from one card to another card. Select a button and chose **Effect...** in the **Button Info** dialog box. This will give you a Visual Effects dialog box containing a large number of visual effects. A complete list of the available visual effects is given in Appendix B.



You will want to experiment with these visual effects and begin to form some ideas about how they contribute to creating effective communication. For example, suppose you are using a left-pointing arrow button to move to the previous card. Try the "scroll left" and "scroll right" visual effects with this button. Which seems better for adding the idea of movement to the previous card?

The following graphic shows the scripting for a "scroll right slow" specification of a visual effect on a button named Next. The third line of the script reflects the change discussed in the previous section that introduced the idea of scripting a button. By imitating this example, a user of a version of *HyperCard* earlier than 2.0 can add visual effects to buttons.


File Edit Go Script
Script of card button id 1 = "Next"
on mouseUp
visual effect scroll right slow
go to card "Demo2"
end mouseUp

People who are first learning *HyperCard* tend to get a great deal of enjoyment out of using the visual effects. They may place a visual effect on each button they create. Typically this is done with little or no consideration about how the visual effect aids in communication or detracts from effective communication. It may well be that a very slow dissolve seems appealing to you the first few times you use it. However, you and others using your stack may eventually find that this wastes a lot of time, is distracting, and adds nothing to the communication. The issue of when and how to use visual effects will be discussed in more detail in Chapter 9.

The speed of visual effects varies depending on the machine you are using. If you develop your stack on a Macintosh Plus and then run it on a Macintosh SE/30, you will see a great difference in the speed of your visual effects. Be sure to design your stacks for the machines on which they will eventually run.

Consistency

As you create a *HyperCard* stack, keep in mind that you are creating a document designed to communicate with yourself and/or others. The stack should be designed so that it is easy to "read" (navigate through).

One design characteristic that is especially valuable to the stack user is consistency. If a left-pointing arrow button is used one place in a stack to mean "move to the previous card," then the left-pointing arrow should mean that throughout the entire stack. If a question mark icon button

means Help at one place in the stack, it should mean that throughout the stack.

Consistency is also important in button placement. Suppose, for example, that you have decided to use left and right arrow icon buttons to link to the previous and the next card, respectively. Then whenever you use this pair of buttons in a stack, they should always be located in exactly the same place on all cards. Remember that **Copy Button** and **Paste Button** can aid you in placing buttons in the same location on each card.

Many people simply use the default round rectangle button for every button they create. Using a wide variety of button types can make your stack easier to use and understand. Techniques for using buttons to communicate more effectively are discussed in more detail in Chapter 9.

A Button Is an Object

The *HyperCard* system makes it very easy to create a button and to give the button a variety of different properties. If you copy and paste the button, these properties are carried along with the button.

The idea of an object (in this case, a button) that can have a number of properties is a very powerful idea in computer programming. You can see some of this power when you copy and paste a button with a large number of properties. The *HyperCard* system provides the programmer with five different types of objects. Some programming languages provide the programmer with a great many more types of objects and also with the ability to easily create additional types of objects. These languages are called object-oriented programming languages.

Nonsequential Communication

A button provides a way for a *HyperCard* stack to interact with the stack user. Buttons facilitate the development of nonsequential communication that is somewhat individualized to the needs of different readers. By clicking on buttons, the reader indicates decisions about what paths to follow through a *HyperCard* stack.

When communicating in "ordinary" sequential written text, the writer typically assumes that the reader will start at the beginning and proceed linearly page by page until reaching the end. Thus, discussions near the end of the document can assume that the reader has read the first parts of the document.

Such is not the case with nonsequential documents. Typically there are many paths that can be followed through the document. Different readers will follow different paths. Thus, by the time the various readers near the end of the document, they may have covered substantially different sets of

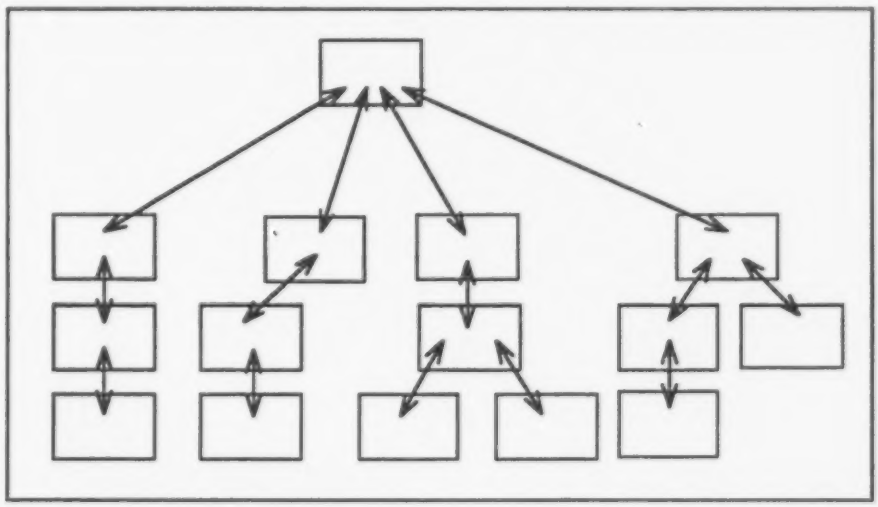
material and developed substantially different types of understanding of the material. Needless to say, this is a challenge to the stack designer!

Detecting and Correcting Bugs

A mentioned previously in this book, a bug is an error in computer software or computer hardware. By now, you most likely have encountered a number of bugs of your own making in the *HyperCard* stacks you have created.

For example, in using the Paint tools, you have probably drawn graphics that were not at all like you had pictured in your mind's eye. ("I wanted to draw a circle inside a rounded rectangle. Instead, I ended up with an ellipse inside the rounded rectangle.") This type of bug is usually easy to detect ("It doesn't look right.") and to correct ("The **Undo** and **Revert** commands and the **Eraser** tool are certainly useful!").

Some bugs are difficult to detect. Suppose, for example, that you are creating a stack with the cards linked as shown in the following diagram. This might be a family tree.



In your implementation of this stack map, each double arrow might correspond to two buttons, Thus, the stack contains a great many buttons. It would be possible—but tedious—to fully test all buttons in the stack, if one were careful and systematic enough.

However, a person often works on a stack over a period of hours or days. Cards are added, moved to different locations, and deleted. It takes a lot of effort to fully test every button in the stack. It frequently happens that you believe you have created a bug-free stack, but you might have added bugs since you "fully debugged" your stack or you did not really fully debug your stack to begin with.

Beginning users of *HyperCard* tend to create stacks containing lots of bugs, and they tend to be relatively poor at detecting and correcting bugs. Skill in detecting and correcting bugs comes through practice and careful thought. One's expertise grows through experience. Moreover, quite a bit of transfer of learning occurs between different programming language systems. A person who is skilled at detecting and correcting bugs in a programming language such as BASIC or Pascal will probably find that this is a big help in detecting and correcting bugs in a *HyperCard* stack.

Problem Solving

You now know how to create a new stack containing a number of cards, each with graphics on them. You know how to create buttons that link the cards. You know how to add visual effects to the buttons. In other words, you know enough to make a royal mess!

Think of a stack as a document intended to communicate with yourself and/or others. It has a purpose—it is designed to communicate a particular message or help the user to solve a particular problem.

For example, suppose you have posed a certain task to yourself ("I want to create a stack that helps the user learn about dinosaurs"). This creates a problem situation that consists of three major parts:

1. Is the stack bug-free?
2. Does the stack accomplish its intended purpose?
3. Is the stack well designed? For example, is the stack easy to understand, are the individual cards well designed, is the stack easy to modify, does the stack "hang together" in its uniformity of appearance, and so on?

The previous section discussed detecting and correcting bugs. Debugging can be a difficult and time-consuming task. Indeed, it is very important to learn to avoid creating bugs in stacks. Usually it is less work to avoid creating bugs than it is to detect and correct them.

Suppose that you have fully debugged your dinosaur stack. The next question is, how can you tell if your stack accomplishes its purpose? Your stack is a communication tool—most likely using nonsequential hypertext. Typically, the best way to tell if a stack accomplishes its purpose is to test it with a group of users for whom the stack is intended. If the stack is intended to be used by third-grade students, it needs to be tested carefully with third-grade students. Kids are more likely to find the things that are "flaws" from their point of view.

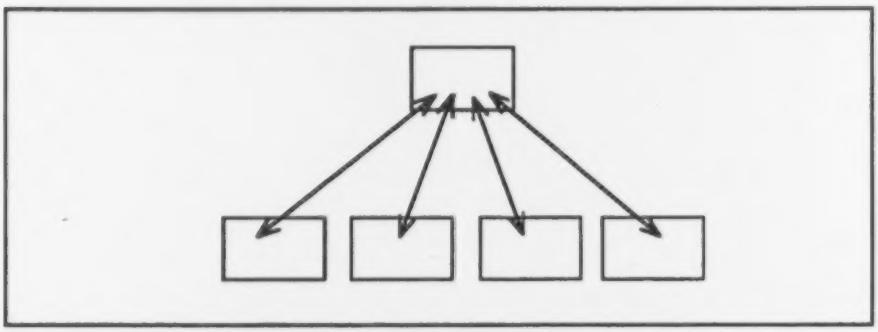
Of course, some people are much better than others at putting themselves in other people's shoes. Perhaps you are very good at functioning like a third grader. If so, you might be able to do the first level

of testing of your stack and make an initial determination of how well it accomplishes the task for which it was designed. Typically, however, you will need to have others use your stack in order to determine how well it actually works.

Activities

1. Experiment with each of the button styles. Experiment with each of the visual effects. Select a button style and a visual effect, and give an example of where this pair of choices would be particularly effective in a stack. Then select a different style and visual effect and give an example of where this button and effect are particularly useful in a stack. Do this several more times.

2. Pose a problem that can be solved or a task that can be accomplished by using a five-card stack with the following stack design. For example, the first card might contain the names of four different vacation spots, while the subsequent cards contain graphics and/or descriptions of the vacation spots.



   Develop the stack in *HyperCard*. Create and fully debug the five-card stack.

3. This is a continuation of Activity 2. Make a list of at least three other distinctly different problems that can be solved or tasks that can be accomplished using the stack design given above. Then discuss the value of creating a stack that is a "template" consisting of five cards linked as above, but not containing any of the details on how to solve a particular problem or accomplish a particular task.

4. There are lots of different tasks you can accomplish using your current knowledge of *HyperCard*. A few examples are given in the following list. As you learn more about *HyperCard* in subsequent chapters, you may want to return to these same activities and redo them.

   A. Create an interactive, fully illustrated, branching story.

   B. Create a story containing buttons that the user clicks on to get graphics that accompany the story.

C. Create a timeline. This is a line covering a period of time, with different events on the timeline. Each event could be a button leading to a card giving more detail about the event. Indeed, a destination card could contain a more detailed timeline with its own set of buttons.

D. Create a piece of an illustrated dictionary or encyclopedia.

E. Create an illustrated story that is available to the user in more than one language. The illustrations can be the same in both languages.

F. Create a multiple-choice or true-false quiz.

G. Create your own collection of clip art, with buttons leading to the cards containing individual pieces of art. Experiment with making transparent buttons that you place on top of a graphic. Through the use of this technique, you can create buttons that appear to have icons you have created.

5. Create a stack that is designed to accomplish some task you have in mind and that is to be used by others. Test the stack on several of your colleagues. Use their feedback to improve your stack.

6. Create a stack designed to accomplish some task. Fully debug the stack. Then insert one or two bugs into the stack, give the stack to a colleague, and tell your colleague that the stack contains bugs. Your task is to report on how your colleague goes about detecting the bugs and whether your colleague is able to find all of the bugs.

7. List some of the techniques you and/or your colleagues use to detect bugs in a stack. Then discuss the applicability of these techniques to detecting errors you make in attempting to solve problems when you are not using *HyperCard*. (This is a High Road transfer exercise. If possible, give examples derived from your use of other pieces of computer software and from areas not involving computers.)

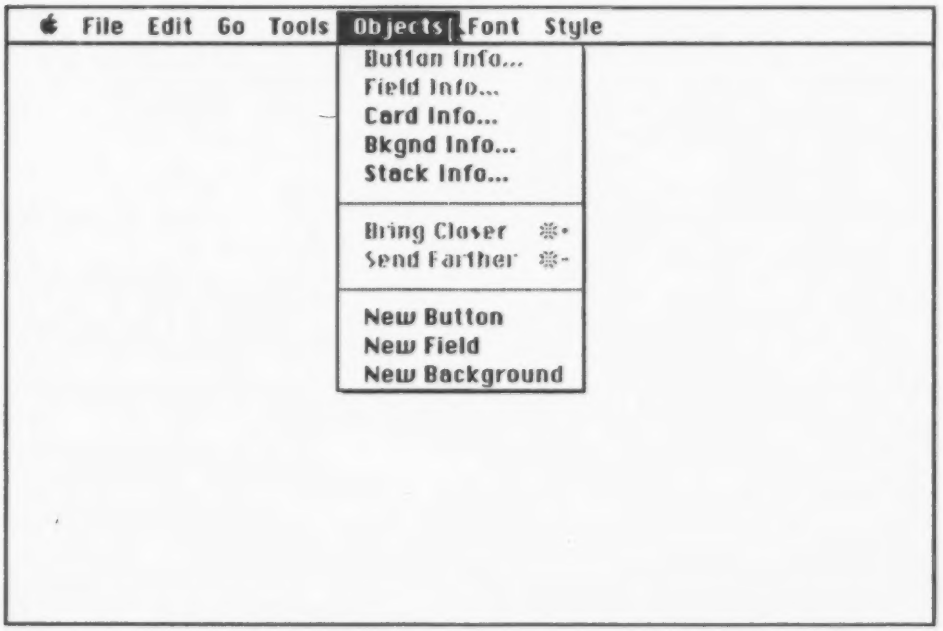
Chapter 7. Fields

A field is another of the five types of objects in the *HyperCard* system. This chapter covers creation and use of fields. Fields are a flexible aid to placing text on cards, and text in a field can be edited just like editing using a word processor.

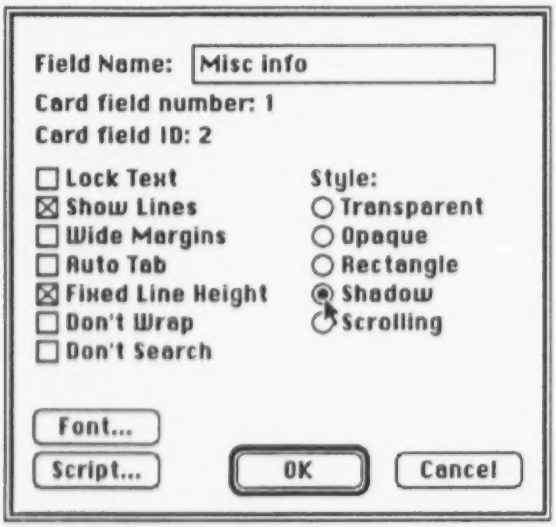
You will learn that using a field is usually a better way of putting text on a card than using the **Text** tool from the Tool window. That is, when you face the problem of placing text on cards, you will be able to solve it using the Paint tools or by using a field. In most cases, use of a field will turn out to be the better method of solving the problem.

Creating a Field

To create a field, you follow steps similar to those you used when you created a button. Begin by creating a new stack or by opening a stack that you use for trying out new ideas. Choose **New Field** from the **Objects** menu.



Then choose **Field Info** and decide on the design of your field.



Technical Note: If you are using a version of HyperCard *earlier than 2.0, the dialog box will look slightly different.*

Entering Text Into a Field

Next, make your field the size you want by clicking near the corner of the field and dragging. You can then place your field wherever you want by clicking within the field and dragging it to the location of your choice. Finally, use the **Browse** tool to place text in the field. Note that the "hand" changes to an "I-beam" when it enters the field. As you begin to type, the pointer changes to a vertical bar.

Previously in this book you "painted" text onto a card by using the **Text** tool from the Tool window. Such text is part of the graphics on the card. It can only be changed by using the Paint tools, such as the **Select** tool and Delete key, or the **Eraser** tool. However, the text you have just entered in the field can be edited like the text created using a word processor. Try placing the pointer in the middle of the line you just typed and adding some words. You see that you can easily modify the text in a field.

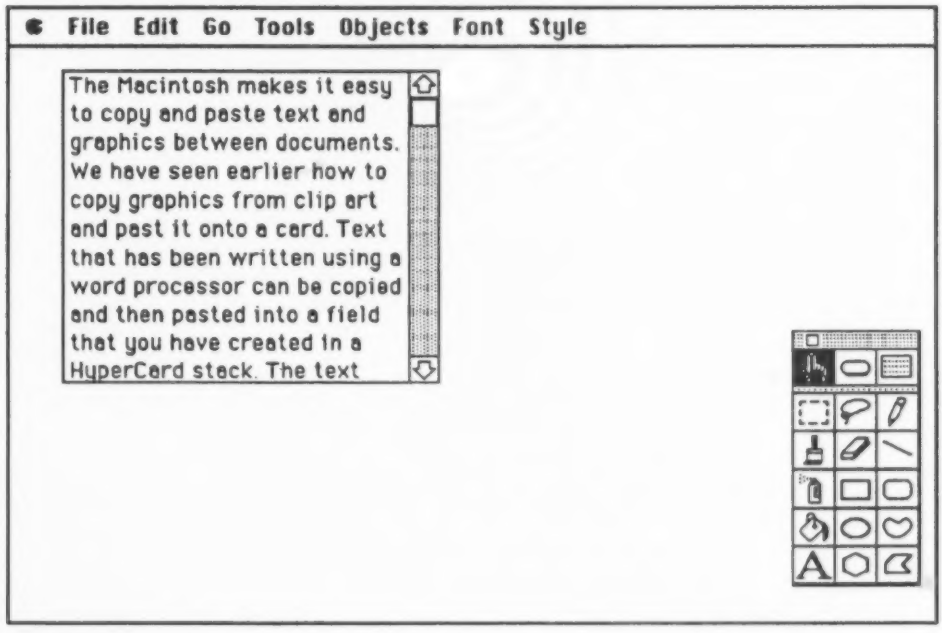
Field Properties

A field is an object, in the same sense that a button is an object. You can specify a number of different properties of a field. You can then copy and paste the field, and these properties are carried along with the field. For example, suppose you want a number of cards to each have a particular scrolling field, and the field is to be placed at the same location on each card. Create the field on the first card, give it the properties you want, and then use the mouse to give the field the size and location you want. Then

use **Copy Field** and **Paste Field** from the **Edit** menu to paste copies onto other cards.

Cutting and Pasting From Other Documents

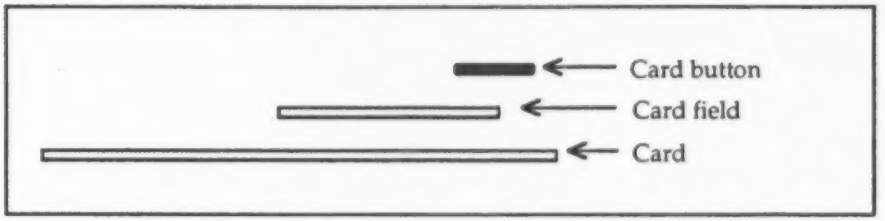
The Macintosh makes it easy to copy and paste text and graphics between documents. We have previously seen how to copy graphics from clip art and paste them onto a card. Text that has been written using a word processor can be copied and then pasted into a field that you have created in a *HyperCard* stack. The text will "flow" into the field, as shown in the following illustration that makes use of a scrolling field (notice the scroll bar on the right side of the field).


File Edit Go Tools Objects Font Style
The Macintosh makes it easy to copy and paste text and graphics between documents. We have seen earlier how to copy graphics from clip art and past it onto a card. Text that has been written using a word processor can be copied and then pasted into a field that you have created in a HyperCard stack. The text

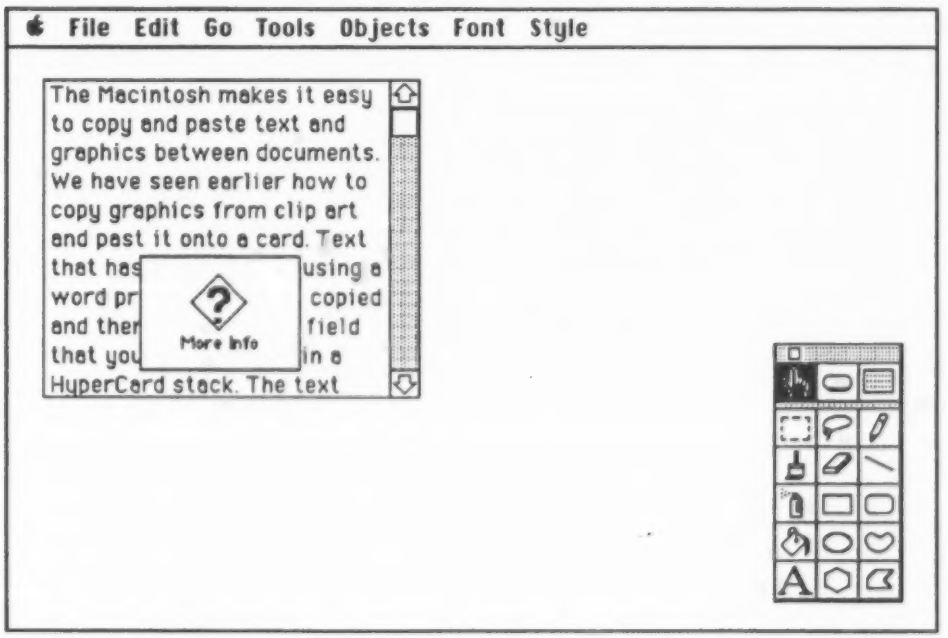
Objects Are Layered

You now know how to create both buttons and fields, and you know how to place them wherever you want on a card. What happens if a button is placed on top of another button or on top of a field? What happens if a field is placed on top of another field, or on top of a button?

The following two diagrams show views of a button created after a field has been created. The button can be on top of the field, which is on top of the card.



It is important that you think about this idea of "layered" objects. In the following example, the button appears to be on top of the field.


File Edit Go Tools Objects Font Style
The Macintosh makes it easy to copy and paste text and graphics between documents. We have seen earlier how to copy graphics from clip art and past it onto a card. Text that has using a word pr copied and ther field that you in a HyperCard stack. The text
More Info

You will want to experiment with different types of situations like this. For example, in the above card, what happens as you scroll down the text in the field? Does the button also scroll? (No, it does not.)

In discovery-based learning, it is important to first form a conjecture and then test your conjecture. If your conjecture is correct, then testing the conjecture and verifying its correctness reinforces your insights that led to the conjecture. If your conjecture proves to be incorrect, you should give careful consideration to what led to the incorrect conjecture.

The layering of buttons and fields on top of each other is affected by the order in which the buttons and fields were created. As you can well imagine, it is easy to create quite a mess that is confusing both to you and to users of the stack you are creating. Strive for simplicity of design. One characteristic of a good design is that the stack is easy to understand and edit.

Pop-Up Field

Suppose you want to have a card that contains some pictures so that when stack users click on a picture, they get some information about the picture. For example, perhaps the pictures include the following clip art ape and raccoon.

A solution to this problem requires scripting. First, create two transparent buttons. Name one of them Ape Button and the other Raccoon Button. Make sure that Show Name is not checked. Size the Ape Button so that it covers the ape; size the Raccoon Button so that it covers the raccoon.

Next, create two fields. Name one of the fields Ape and name the other field Raccoon. Write some information about apes in the Ape field, and write some information about raccoons in the Raccoon field.

Next, put the following script on the button named Ape Button:

```
on mouseUp
   set the visible of card field "Ape" to true
end mouseUp
```

Then place this next script on the button named Raccoon Button.

```
on mouseUp
   set the visible of card field "Raccoon" to true
end mouseUp
```

These scripts will cause the Ape field and the Raccoon field to be displayed (to become visible) when the user clicks on the ape or the raccoon.

Now, all that remains to do is to make the Ape and Raccoon fields invisible so that they can become visible (pop up) when the user clicks on the appropriate picture. To do this, create two more transparent buttons with Show Name not checked. The first one, named Ape Invisible, should exactly cover the Ape field. The second one, named Raccoon Invisible, should exactly cover the Raccoon field. Script these two buttons as follows.

```
on mouseUp
   set the visible of card field "Ape" to false
end mouseUp

on mouseUp
   set the visible of card field "Raccoon" to false
end mouseUp
```

The first of these scripts will cause the Ape field to disappear when the user clicks on the Ape Invisible button—that is, any place on the Ape field. The second of the two scripts will cause the Raccoon field to disappear when the user clicks on the Raccoon Invisible button—that is, any place on the Raccoon field.

Finally add to the Ape and the Raccoon fields directions to the user, such as "Click on this field when you are through reading it." Put some general directions on the card, such as "Click on a picture to get more information."

The pop-up field uses buttons layered on top of graphics and fields. In your initial exploration of the pop-up field, make sure that the field that is popped up does not overlap the graphic that it explains.

Problem Solving

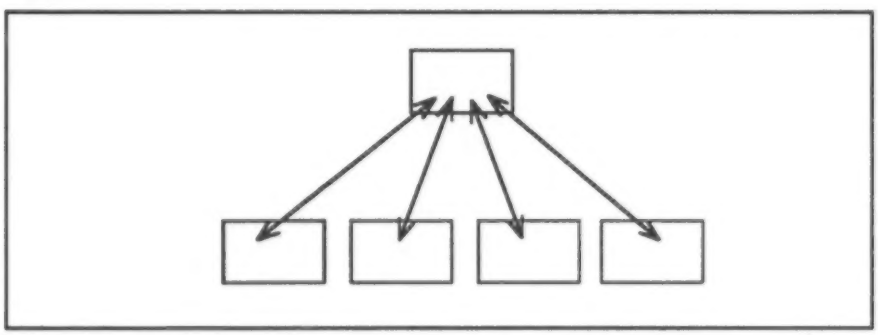
Take some time to reflect on what you are learning about *HyperCard* and how you are learning it. Each chapter in the text introduces you to one new set of *HyperCard* capabilities. First you learned to browse through a stack that someone else had written. Then you learned to create a stack consisting of several cards. Then you learned about the Paint tools. Then you learned about buttons and their properties. Now you have learned about fields and their properties. You have learned how to script a button; one use of this scripting is to create a pop-up field.

Each new chapter adds three difficulties. First, there is the actual new material to be learned. Second, there is the interaction of this material with previously learned *HyperCard* material, as well as with your other knowledge. Third, there is the growing complexity of problems that you can address using your growing knowledge of *HyperCard*. You now know enough about *HyperCard* to attempt some really complex tasks.

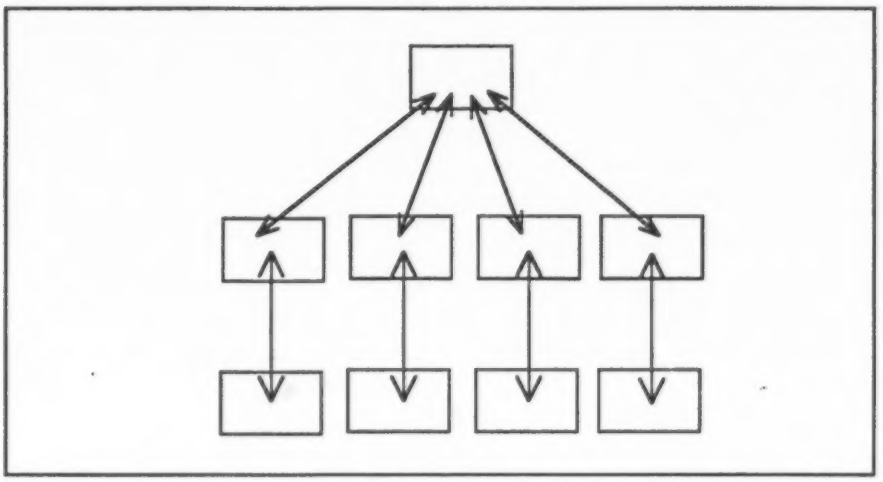
Problem posing—defining a problem to be solved or a task to be accomplished—is a very important part of problem solving. In learning *HyperCard*, you want to pose problems for yourself that are just a little harder than the ones you are currently comfortable in solving. And remember, persistence is very important in problem solving. Thus, pose problems for yourself that are challenging and that take persistence.

Activities

1. Examine the following stack map and imagine that each of the four cards in the "second row" contains a field. Think of a problem that can be solved or a task that can be accomplished by using this type of five-card stack. For example, the first card might contain the names of four different restaurants, while the other cards provide information about the locations and menus of the restaurants. Develop this stack in *HyperCard*.



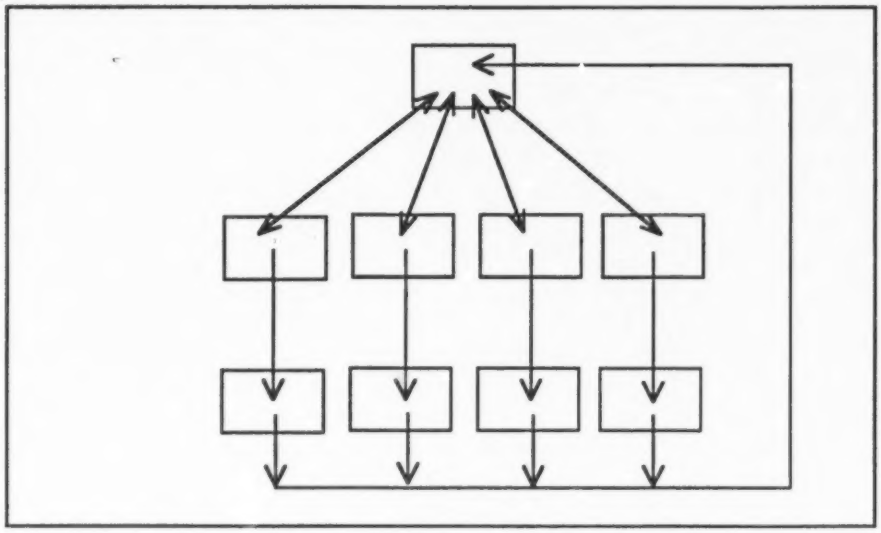
2. This is a continuation of Activity 1. In Activity 1, each of the four cards in the "second row" contains a field that contains some information. This information might be divided into two parts, where a button on the card is clicked to move to additional information on the topic. The following illustration shows a stack map for this process.



Pose a problem that can be solved with this stack design and then develop a *HyperCard* stack for it. If you are using the restaurant example, the "second row" of cards might be sample menu items, while the "third row" might contain maps showing how to get to the restaurants.

3. This is a continuation of Activities 1 and 2. In Activity 2, two cards were used for each set of information that is accessed from the card in the first row. An alternative would be to use the stack design given in Activity 1, but to use a scrolling field instead. A message to the reader could be used to divide the first part of the text from the second. It might say something like "Scroll down if you want more detailed information." Discuss the relative merits of these two designs. Give examples of where one might be more appropriate than the other.

4. This is a continuation of Activities 1, 2, and 3. Examine the following stack map. Notice that the return paths from the "third row" of cards bypass the "second row" of cards. Discuss the advantages and disadvantages of this. Give examples where one might be more appropriate than the other. Develop a stack for one of these examples.



5. This is a continuation of the previous activities. In the previous activities, we referred to the "second row" and the "third row" of cards. Select one of the stacks you have created in Activities 2-4. Start at the first card and go through the cards sequentially using the **Next Card** option in the **Edit** menu, or use the equivalent keystroke command. Then explain why we put "second row" and "third row" in quotation marks. Your explanation should make a clear distinction between the way the computer "thinks" the cards are sequentially ordered inside the computer and the way you think they are arranged according to the stack map you used to design the stack.

6. Experiment with pop-up fields and layering. Pose several different problems for yourself and then solve them. For example, perhaps your card contains a map with names of countries. Each country has a pop-up field that gives more information about the country. How will you deal with the problem that buttons are rectangular in shape but most countries are not?

7. Consider Activity 1, but this time think of having just one card with four pop-up fields. Give an example of a problem where using pop-up fields might be more appropriate than using multiple cards. Develop a one-card stack for your example.

Chapter 8. Backgrounds

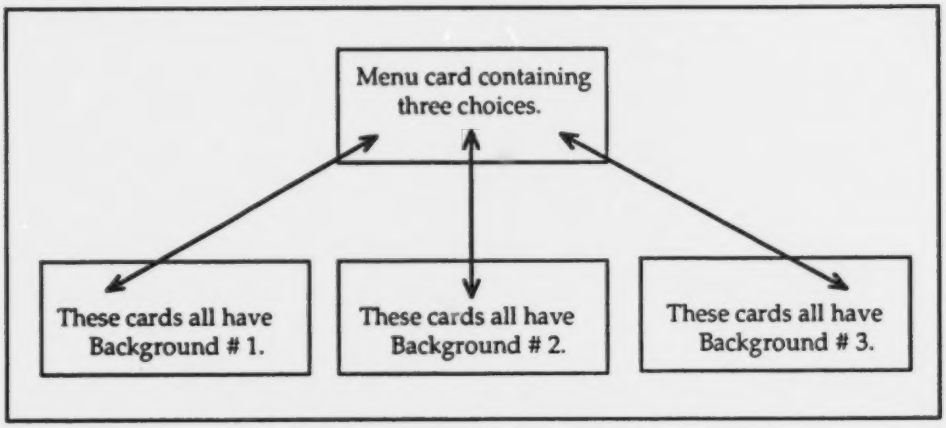
HyperCard provides the user with five types of objects. Previous chapters have covered four of them—stacks, cards, buttons, and fields. This chapter introduces you to *backgrounds*, the fifth of the five types of objects.

Every *HyperCard* card has a background. Up to now in this book, the backgrounds have been blank. In this chapter you will learn how to create nonblank backgrounds. These are a useful design aid for effective communication.

Using Backgrounds

One way to think about background is to think about a stage production. Each act of the production has several different scenes, and each of the scenes uses a different stage setting—a different background. In this analogy, the cards in the stack are the actors and the action. All of the cards in one "scene" have the same background. A change of background is like changing to a different scene in the stage production.

Suppose, for example, that you are going to create a stack containing information about elementary school curriculum, middle school curriculum, and secondary school curriculum. Conceptually, you are thinking of this stack as consisting of three distinct components. Some people using the stack will only want to read through one component. In this case, a design such as that shown in the following illustration might be appropriate.

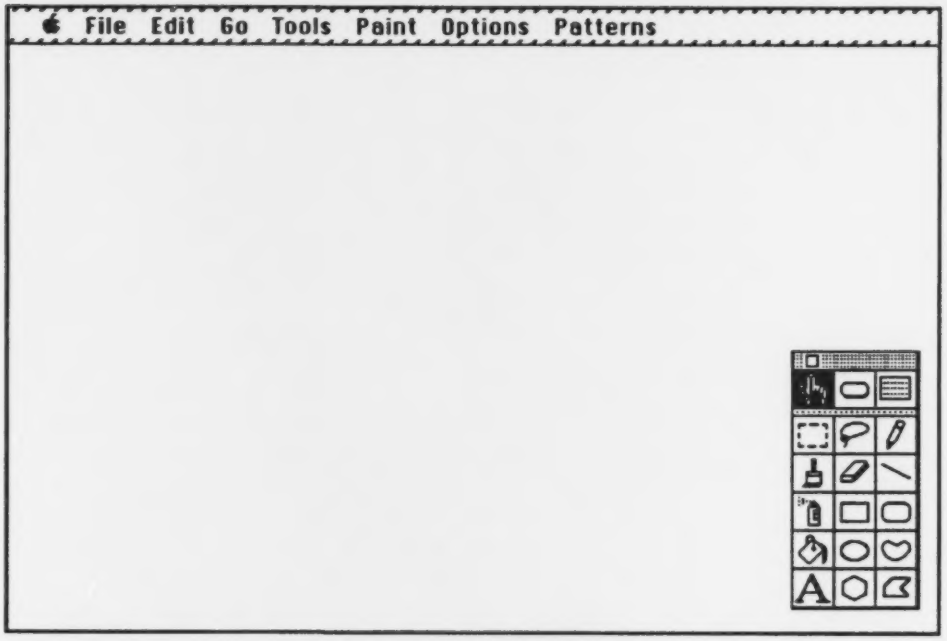

Menu card containing three choices.
These cards all have Background # 1.
These cards all have Background # 2.
These cards all have Background # 3.

Creating a Background

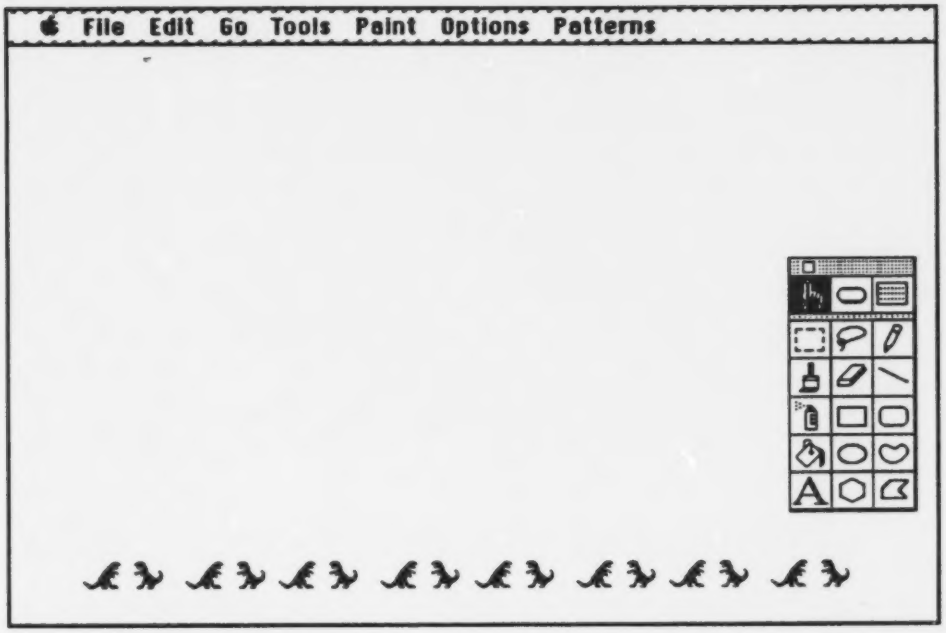
Open a stack you use for experimentation, or create a new stack containing several cards. Select the **Background Info** option under the

Objects menu. You'll see that the background is shared by all of the cards in your stack. In the first part of Chapter 4, we discussed the actions to take so that the background in your stack would be blank. Here we are assuming that the background for your stack is blank.

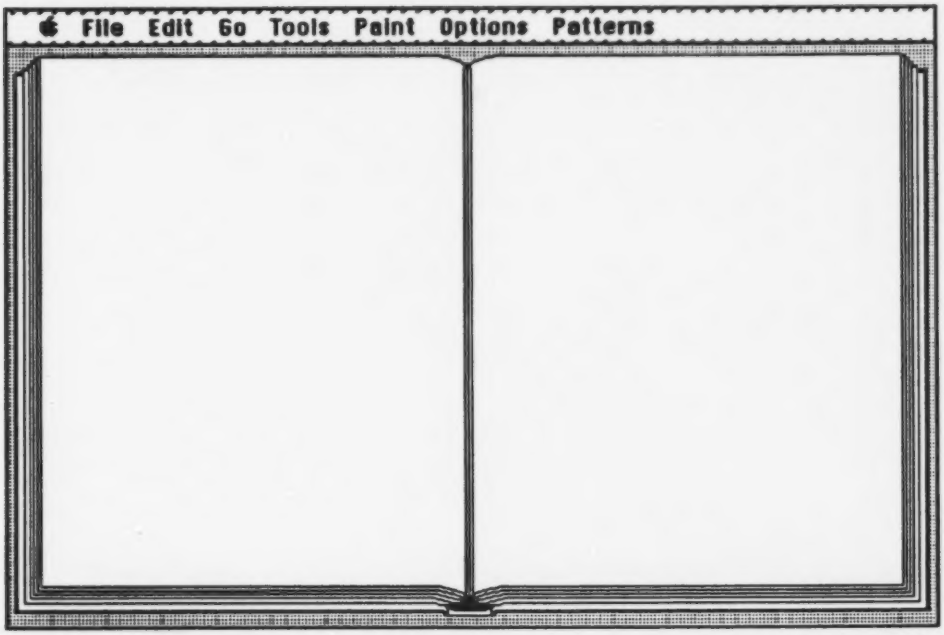
Now you are ready to modify the blank background. Select **Background** from the **Edit** menu. You see small diagonal lines along the Menu bar at the top of the screen, and the rest of the screen will be blank or will show the Tools window if it was showing on the card you were viewing when you selected **Background** from the **Edit** menu.



Whatever you put on this background will become the background for your entire stack. Create a background you can easily recognize. For example, the following background is one that could be used in the dinosaurs part of a stack about animals.



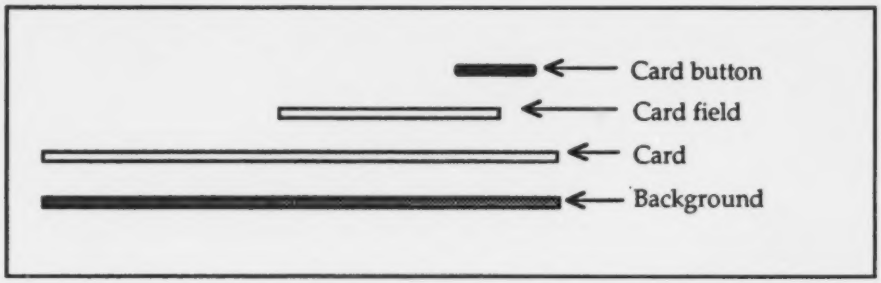
It is common to use clip art for backgrounds. The small dinosaurs used in the background came from clip art. Here is another clip art background that accompanies one of the versions of *HyperCard*. It might be used in a stack that contains a story, where you want readers to feel like they are reading a book.



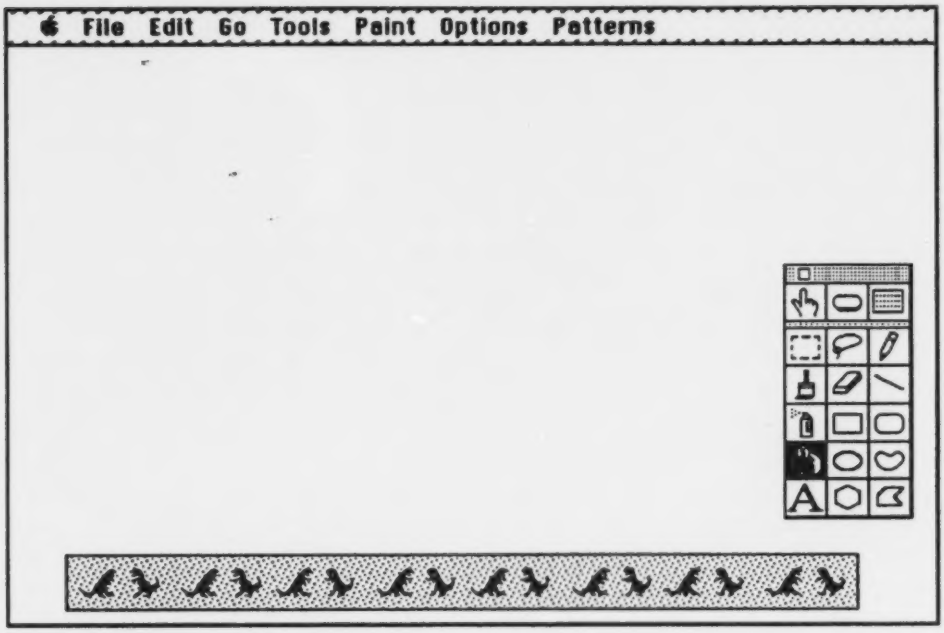
After you have created a background to your liking, select **Background** from the **Edit** menu, or use the ⌘-B (Command-B) keystroke to return to

the card from which you started. You will see that this card and the rest of the cards in your stack all have the background you have created. Remember, in this case all the cards in your stack previously shared the same blank background. You changed this blank background. Later in this chapter you will learn how to create a different background and cards having the different background.

In the previous chapter we presented an edge view of the layering of a button and a field on a card. The following edge-view diagram shows the background sitting below, or underneath, the card button, card field, and card.



A background you have created can be edited in exactly the same manner as the blank background. When viewing a card that has a background you want to edit, select **Background** from the **Edit** menu, or use the ⌘-B (Command-B) keystroke. You can then use the Paint tools from the Tool window to edit the background. The following background was created by drawing a rectangle around the row of small dinosaurs in the previous background, and then using the **Bucket** tool to fill the rectangle. Remember, the window of Paint tools is not part of the background or of the cards in the stack.



New Background

All of the cards in your stack have the same background. This section explains how to create a different background for part of a stack. When viewing a card, select **New Background** from the **Objects** menu. This will give you a blank background. Create a background pattern of your choice, and then use the **Background** choice in the **Edit** menu or the ⌘-B (Command-B) keystroke. By this process you have created a new blank card *with* the new background. None of your previous cards or their background has been changed.

If you were viewing the second card in your stack when you created this new background and new blank card, you will find that the new card is the third card in your stack. Suppose that before creating the new card, the second card of your stack contained a button linked to the third card of your stack. The process of creating a new background created a new third card, and your original third card has become the fourth card in your stack. The button linking the second card to this (fourth) card works just as you would expect—that is, the button links to the specific card, which now happens to be the fourth card in the stack.

It is easy to create additional cards that have a particular background. One way is by using the **Copy Card** and **Paste Card** options in the **Edit** menu. The background is carried along in this copy-and-paste operation.

Alternatively, suppose that you are viewing a card with a particular background. You create a new card by means of the **New Card** choice in

the **Edit** menu or by using the ⌘-N (Command-N) keystroke. The new card will have the background of the card you were viewing.

Buttons and Fields on a Background

It is possible to place buttons and fields on a background. The process is essentially the same as placing buttons and fields on a card.

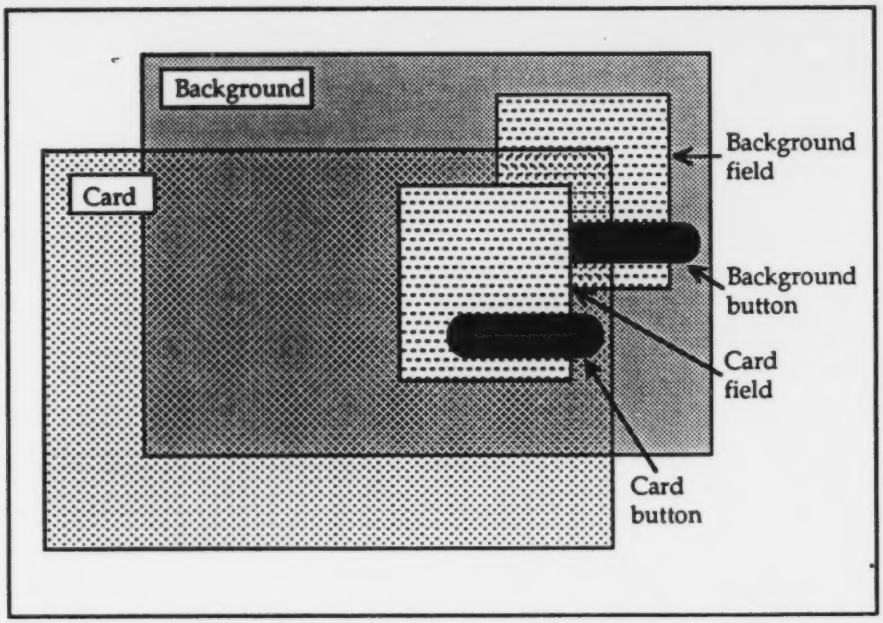
Suppose you are creating a stack that begins with a menu containing several choices. You have decided that each card in the stack will have the same background. In your stack design, you want the user to be able to easily return to this menu from any place in the stack. One solution is to place a button on the background and link it to the menu card. The button has to be placed only once and linked only once.

Of course, that means that the button will also appear on the menu card, and that might be distracting to the stack user. You can cover this button on the menu card by placing an Opaque button on top of it. (Make sure Show Name is not checked.)

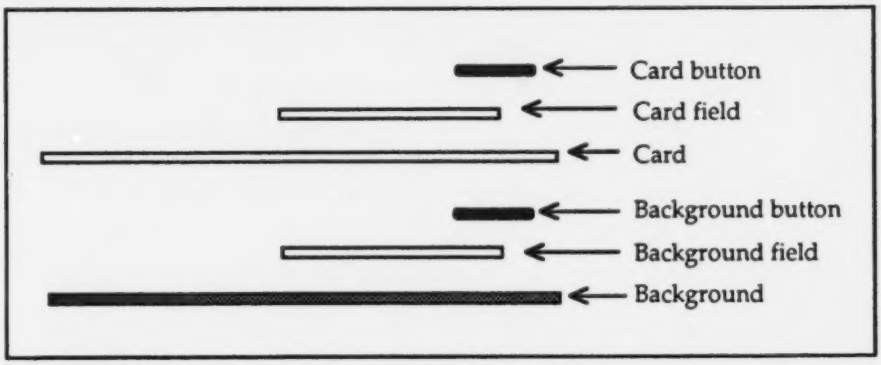
In a similar vein, you might want a particular set of instructions to appear on every card. Put the set of directions in a field on the background.

Layering Buttons and Fields

All the objects on a card or background are layered. That is, you can move some objects "on top of" other objects. If you create a single *HyperCard* card and make one background field, one background button, one card field, and one card button *in that order*, they are layered as shown in the following diagram.



A cross-sectional view of the same diagram is given in the following illustration. A key part of this diagram is that everything on the card is above (on top of) everything on the background.



Technical Note: The layering of buttons and fields also depends on the order in which they are created. The preceding diagrams are accurate only for the instance where a single card field is created, followed by a single card button, and for where a single background field is created followed by a single background button. If more than one button is created on the card, the last button created will be "above" the previously created button. However, card buttons and fields will always *be above background buttons and fields.*

Finding Lost Buttons and Fields

If you create a Transparent button and the Show Name option is not selected, the button will be completely invisible. However, it will work

exactly like a visible button. Similarly, an Opaque button with the Show Name option not selected will be invisible when placed on a blank space—for example, where both the card and its background are blank.

Thus, when you are viewing a card or a background, you cannot tell, merely by looking, where the buttons and fields are. You can locate buttons and fields by using the Button tool and the Field tool from the Tool window. For example, suppose you are viewing a card and you select the Button tool. Each button on the card will be displayed with a dotted rectangular box around its edge.

Communication

You have now studied all five types of objects in *HyperCard*. You now know how to create a nonsequential document of considerable complexity.

Most books and other written documents are designed to be read in a sequential fashion. As people learn to read, they learn to read documents that are laid out sequentially. As people learn to write, they learn to write sequential documents.

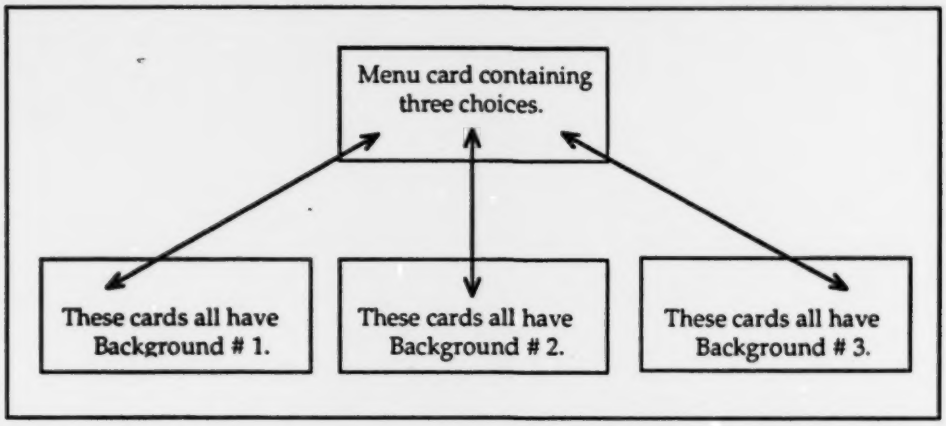
Now you have the tools to write and edit nonsequential documents. These documents need to communicate effectively both with yourself and with others. However, neither you nor your readers are trained in reading (navigating through) nonsequential documents.

It is important to keep these ideas in mind. As you create a *HyperCard* stack, you gain practice in developing nonsequential documents. The reader of these documents gains practice in learning to read nonsequential documents. You want to help this reader/learner as much as possible. Using appropriate visual cues, such as the background, is essential.

Activities

1. Consider the following stack design. The Menu card is to have a blank background. Think of each of the three parts in the second row of this diagram as being a self-contained unit (for example, each could be a stack). Each is to have its own distinct background.

   Pose several distinct problems that can be solved by using this stack design. For example, the stack might contain information about three different sports. In that case, perhaps the three different backgrounds each contain a border of small pictures of players in their sporting uniforms.


Menu card containing three choices.
These cards all have Background # 1.
These cards all have Background # 2.
These cards all have Background # 3.

2. Develop stacks for at least two of the problems you identified in Activity 1. You can save quite a bit of time by working to avoid duplication of effort. Thus, keep in mind that the basic design is the same for both stacks. Think about creating a "template" that can then be "filled in" with the details of a particular problem.

3. This book contains a number of examples of stack maps. Quite a few of the activities suggest that you pose problems or tasks that can be accomplished using a particular stack map. (That is, the stack map is considered to be a template for a particular stack design. Many different problems can be solved using the same template as a starting point.) Discuss the value of including a stack map in a stack so that its user can examine its design. When might this be appropriate and beneficial to the stack user, and when might this be inappropriate?

4. Activity 1 suggests that buttons on a card may be linked to other stacks. Create a new stack consisting of just one card—a menu card—that is linked to several other stacks you have created. Of course, you will want to link those stacks back to your new one-card stack.

5. Suppose you want to have an animated sequence in *HyperCard* in which a cloud slowly drifts above a landscape. The landscape could be a background and the cloud could be placed in slightly different locations on a sequence of cards having this background. Develop and debug such a stack.

6. Suppose you have completed a three-branch stack in which all cards have the same background. You then decide that there should be three different backgrounds, one for each of the three branches. Explain two quite different ways to accomplish this task. Discuss their relative merits, especially in a case where each branch contains a large number of cards.

Chapter 9. Stack Design Issues

In this chapter, you will focus on the design of *HyperCard* stacks. In particular, you will learn more about stack structure, appropriate use of visual effects, and some *HyperCard* conventions you should take into consideration when designing stacks.

To a great extent, the previous six chapters of this book have focused on the mechanics of developing a stack. The stacks you have produced have probably been of limited scope—perhaps best described as experimental and not designed to be a product of lasting value.

You are now ready to begin applying your ideas to building stacks that are of lasting value to you and to others. These stacks must be designed to stand the test of time. Weeks or months after creating one of these stacks, you should be able to return to it, easily use it, and easily modify it. You should be able to give a copy of a stack to a novice *HyperCard* user and expect this person to have no trouble in using it.

HyperCard is a versatile computer application for creating nonsequential documents. It contains many different features for creating a graphical interface to work with information contained in stacks. *HyperCard* is designed to make the creation of stacks quick and easy. However, it is also quite easy to produce very badly designed stacks. The user of such stacks can easily become confused or lost.

Stack Structure

The first step in creating a *HyperCard* stack should be planning the structure of the stack. There are many ways you can put a stack together. One effective approach to stack design is to "outline" your stack on index cards. Using this method, you lay the cards out on a table to help you decide upon the links among cards. You might also consider using a bulletin board to arrange the cards in your stack. You then use pieces of string or yarn to indicate links among cards. This approach lets you take a look at the overall structure of your stack and think about important relationships among cards.

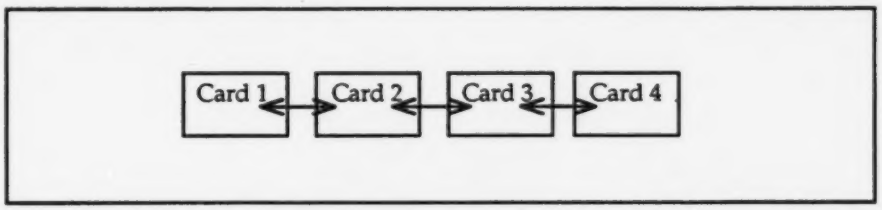
When designing a stack, it is extremely important to think carefully about the nature of the material you want to put into a stack. Is it "linear" in nature? That is, do you want your readers to see one card at a time, much as if they were reading a book? Or does the material "branch" naturally. Is it sensible for the user of your stack to want to explore one area, such as the habitats of animals, for a while and then be able to move with ease to material about the foods eaten by those same animals? Is your stack exploratory? That is, do you want users to be able to explore in a

variety of ways and jump from place to place among the cards in no particular order? Answering questions such as these requires a lot of planning, examination of the material that is to become part of your stack, and thought about the nature of the final product.

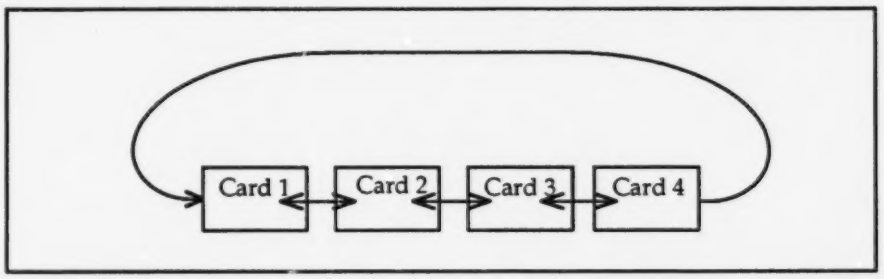
Stacks can have many different formats. Some basic examples are discussed in the following sections. However, it is important that you *not* take these as rules but rather as general ideas about ways to structure stacks. You can create stacks that are exactly like some of the following models, or you can create stacks that are totally different from these models. Or you might choose to use some combination of these models. The structure of your stack depends heavily on the nature of the material that makes up your stack.

Linear and Jump Linear Stacks

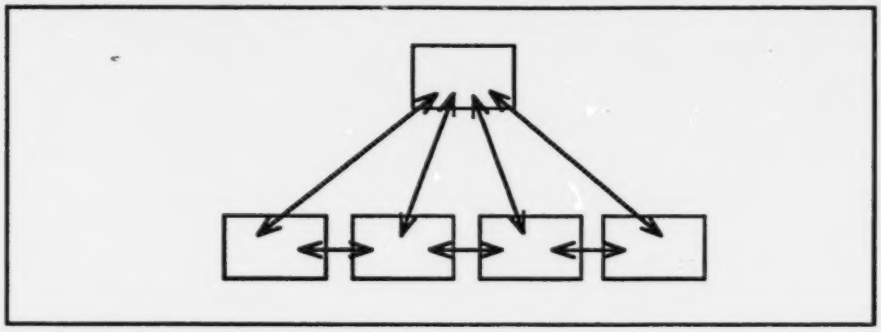
Stacks may be strictly linear. You might use this type of stack if it is important that the user see each card in sequence. This design might be used when you have a sequence of instructions and you want to be sure the user sees each instruction in order. For example, suppose the stack is a set of instructions that absolutely must be done in a prescribed order. All of the instructions on Card 1 must be carried out before beginning to follow the instructions on Card 2.



It is not uncommon for a linear stack to add a link between the first card and the last. This makes it easy for the user to return to the first card without backtracking through the cards after viewing each card in the sequence.



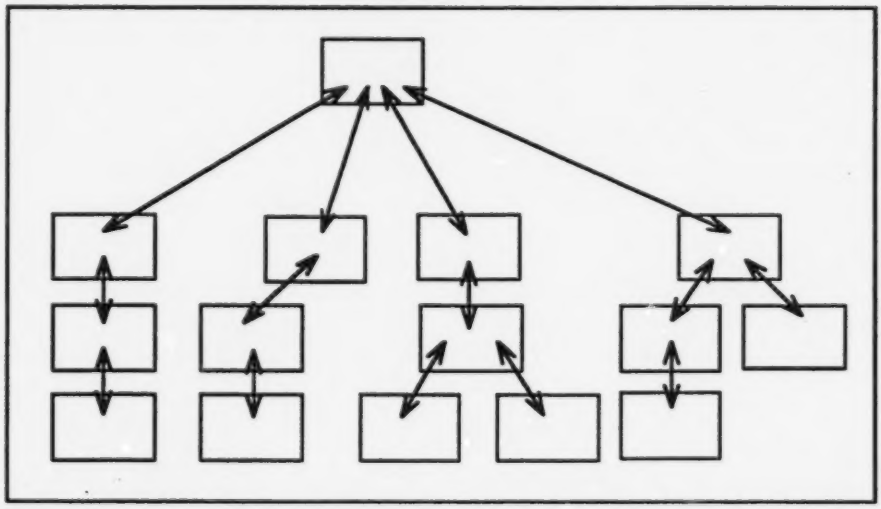
You might also want to make your stack "jump linear," allowing the user to jump into or out of any card in the linear part of the stack.



The jump linear model might be used to present a menu on the first card—the top box in the preceding diagram. The user can then go to any card in the linear sequence. Once in the linear sequence, the user can either go back to the menu or move back and forth in the linear sequence. As in the previous example, the right-most card in the second row might be linked to the left-most card, perhaps with a two-way link. As an example of use of this design, perhaps the menu card contains brief descriptions of four different tourist attractions in a city. The second row of cards contains detailed descriptions of the tourist attractions. A person browsing the stack might want to read about just one tourist attraction. Or the person might want to move from one tourist attraction to another without returning to the menu card.

Tree Stacks

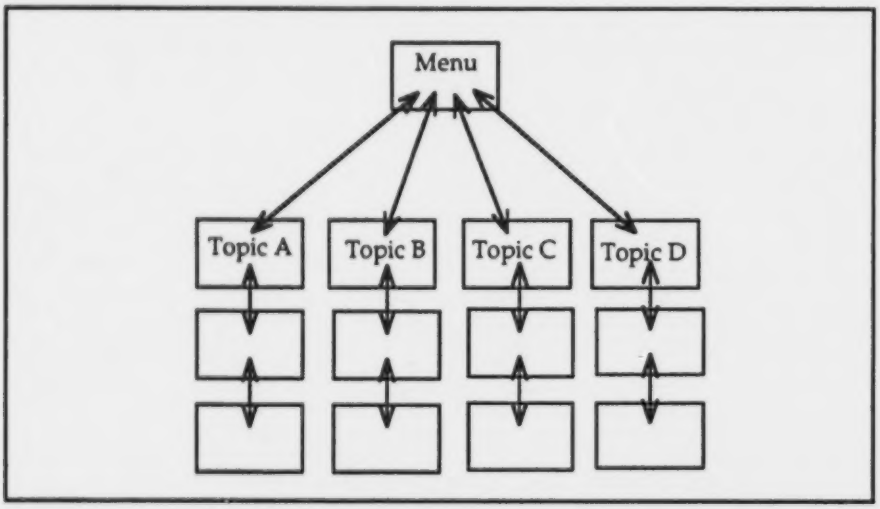
Computer scientists use the term "tree" to describe the structure shown in the following diagram. This is the sort of diagram that can be used to indicate the people in a family tree.



A tree structure allows users to move "up" and "down" branches of the tree to explore any particular idea in more depth. Tree structures are

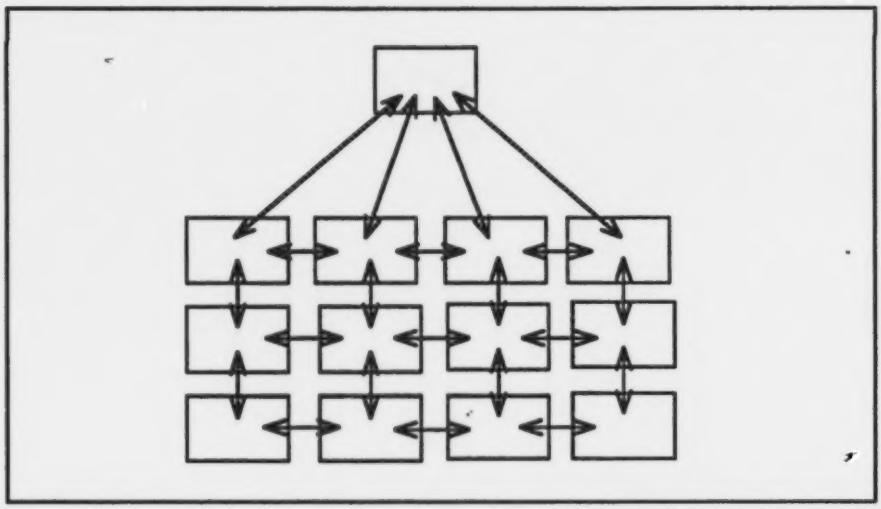
fairly natural ways to display information. For example, if your stack is about animals, you might have two subbranches: water animals and land animals. Then, under water animals, you might have other divisions, such as vertebrates and invertebrates.

There is a temptation when working with tree structures to make as many links as possible among cards. Such an approach can be quite confusing to the user. The following tree structure is fairly easy to follow. Users know that once they are in a given branch of the tree, they will remain in that branch until they return to the top of the tree.



However, suppose that a particular branch might be dozens or even hundreds of cards long. Then it could take the user a long time to traverse back up the branch to get back to the main menu. As an alternative, every card under Topic A might contain a button linked to the first card in the branch, and similarly for the other topics. Each card in the branch might also contain a button linked to the Menu card.

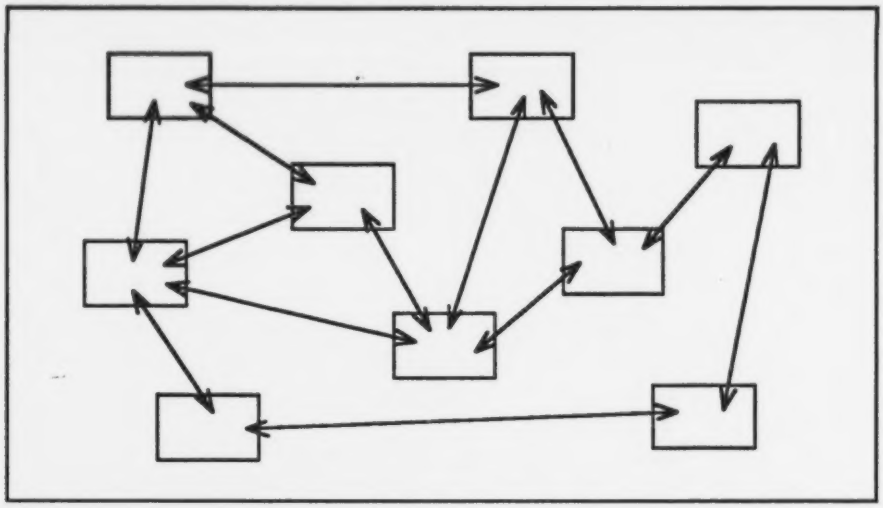
If many links are made that allow movement across the branches of the tree, users can easily become lost. The following diagram shows a structure in which it is easy to become lost.



In this case, assume the stack is about animals. If users go down the "water animals" branch of the tree and then move to the right or left, they can easily find themselves in a "land animals" branch without being aware that the nature of the information being presented has changed. Making as many links as possible does not necessarily make your stack better. However, you need to be sure the links to the branches of a tree are bi-directional; that is, be sure the cards are linked so that a user can in some manner go "down" and then back "up" a given branch of the tree.

Finally, network stacks are used when you want the user to explore your stack in a variety of different ways. Such stacks present information that has no particular order to it. For example, you might represent airline routes or highway routes in such a stack. A button for a city might activate a pop-up field that gives the flying time or driving time to get to a connected city, while a second button actually moves you to that city.

As an example of another use of a network stack, you might create an exploratory stack for children so that a picture of an object contains buttons that move the viewer to a related object.



Single-Frame Stacks

Although the model that we use to think about *HyperCard* stacks is a stack of index cards, not all stacks appear to the user to have a lot of cards. "Single-frame" stacks appear to have only one card. The user has no sense of moving through a number of cards at all. One type of single-frame stack is the "display stack." This type of stack is used to show a graphic or text field that changes depending on the user's choice. For example, you might create an alphabet stack for children. Along the top of the card there would be 26 buttons, one for each letter of the alphabet. When the child clicks on the letter, a picture of an object beginning with that letter appears. In such a stack, the pictures might appear in a standard "frame" on the card, and no visual effect would be used to create a sense of movement.

Another form of single-frame stack is the "filter stack." This type of stack may serve as a kind of menu for a single stack or "front end" for another group of stacks. Filter stacks are particularly useful when a stack becomes very large and contains a lot of information. For example, a stack that contains information about all of the major countries of the world might filter the information by allowing the user to choose to look at, for example, African countries only.

A third type of single-frame stack is a "point-and-click" stack. Such stacks let the user click on various items on the card to learn more about them. For example, a stack that presents a map of a city might give brief descriptions, addresses, and phone numbers of restaurants when the user clicks on the appropriate picture on the map. In such a single-frame environment, the user would have no sense of moving to another card; rather, the information would appear to "pop up" on the original map card.

111

Using Visual Effects

Technical Note: If you are using a version of HyperCard *earlier than 2.0, visual effects must be scripted. Moreover, they will not work if the monitor being used is set to color.*

Visual effects are a lot of fun. However, it's quite easy to end up using visual effects that are inappropriate for your stack just because *you* enjoy looking at them. Perhaps you have already discovered how easy it is to over-use visual effects as you were developing a stack. After you see the same powerful effect many times as you are debugging, you soon find that these effects can be irritating as well as exciting.

Consider the following analogy. Visual effects are like doorways. You put a door (visual effect) in only when you want to make a transition explicit or close off one section from another. You don't put a door between your chair and your television, because it would be in the way. Nor do you put a heavy bank-vault door on your kitchen cabinet when an easy-to-open door serves the purpose much better.

You should use visual effects only when you have a specific reason for doing so. It is not possible to give a definitive set of rules for using visual effects. What you see on the screen can be rather subjective. An effect that appeals to you may annoy another person. Furthermore, the effect that you see on the screen is strongly influenced by the cards involved. A visual effect that is quite strong in the case where the first card is quite different from the second can become quite weak when the first card is almost like the second. Here are some very general guidelines for using visual effects.

Scroll

A scroll makes the next card appear to slide into position from the side or top. From the user's point of view, the "new" card does not appear to have been underneath the first. This effect generally makes users feel as if they are moving along to a new topic. This gives the feeling that a transition of some kind has taken place. Scrolls are also good when you want to simulate a slide show, because the effect on the screen is quite similar to seeing a set of slides projected by a slide projector.

Dissolve

Dissolves give an emphasis to the transition itself. You should use a dissolve when there is a significant graphic contrast between cards or when you are changing one small part of a card and want to draw the user's attention to that change. If the cards are quite different, a dissolve can give the user time to adjust. Dissolves tend to be midway in strength between scrolls and wipes.

Wipe

A wipe gives more of an impression that a page has been turned and that the next card is underneath. The transition produced by a wipe is less abrupt than a scroll. This is a good effect to use if you want users to feel that they are continuing at the same level.

Iris/Zoom

These two effects are quite powerful. They can simulate the feeling of going into or out of three-dimensional space. They give users a feeling of depth, as if they are going more deeply or less deeply into the material being presented.

Barn Door

This effect produces a feeling of finality, like opening or closing a scene. There is no sense of transition as the user moves to the next card. This effect should be used to indicate the beginning or end of a major section of a stack.

Checkerboard/Venetian Blinds

These two effects are very powerful. They should be used only for a strong visual break and should be used sparingly. Many users find them annoying.

HyperCard 2.0 Effects

The *HyperCard* 2.0 effects *shrink* and *stretch* are not available in earlier versions of *HyperCard*. They are similar to scrolls when used with "to top" and "to bottom." They are more like irises and zooms when used with "to center." However, in both cases these two effects are somewhat more dramatic than their scroll and iris/zoom counterparts.

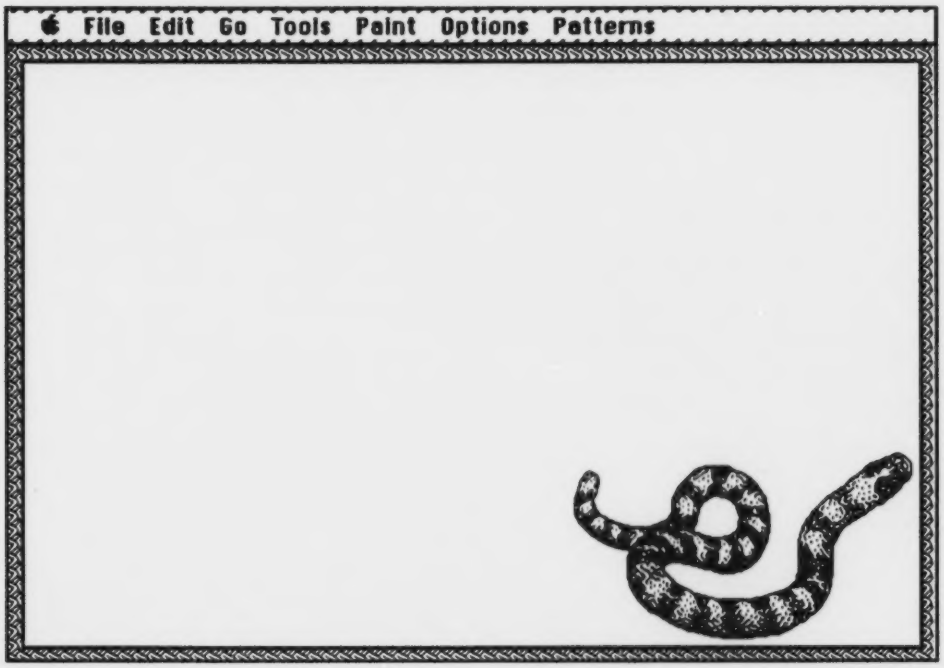
Backgrounds

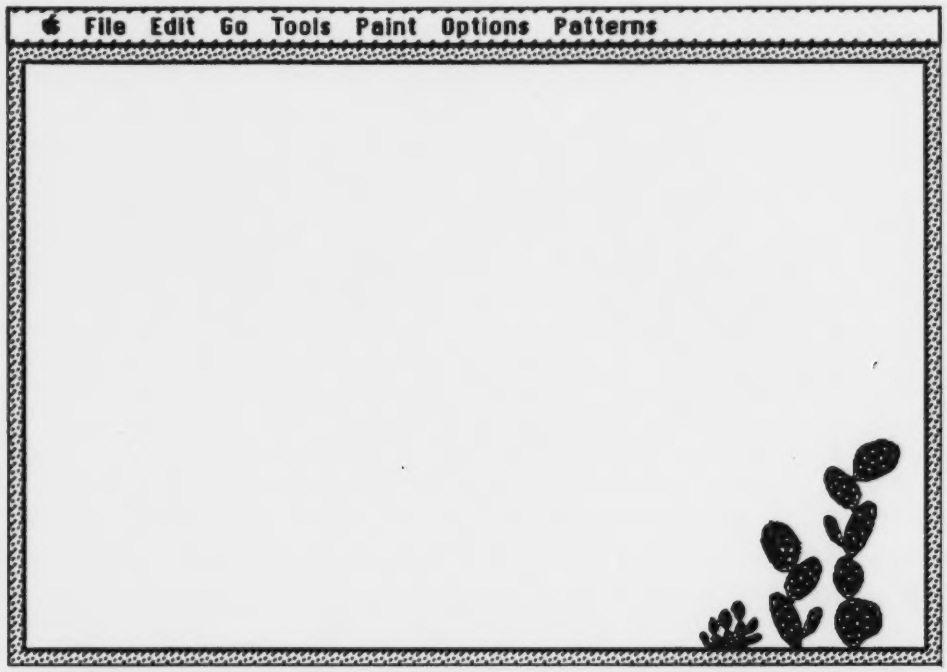
Some beginning *HyperCard* users seem to use the same background all the time. They simply take the background *HyperCard* puts into a **New Stack** and use it for every stack they create. Others, however, discover that they can put any kind of picture on the background. They end up with a different background on every card. Still others put "next" and "previous" buttons on every card None of these approaches makes appropriate use of *HyperCard* backgrounds.

The initial background you choose for your stack should reflect the stack's content. It certainly is inappropriate to have a background that looks like index cards if your stack is supposed to look like a slide show.

You should give a good deal of thought to the backgrounds you design for your stack.

Because it is so easy to edit and change backgrounds, it is tempting to have a lot of backgrounds in your stack. However, this can be very confusing to the user of your stack. A background change should represent a change in the material being represented in your stack. For example, if you are teaching about desert animals and plants in your stack, you should have at least two backgrounds: one to indicate that the user is in the section about desert animals and one to indicate that the user is in the section about desert plants.





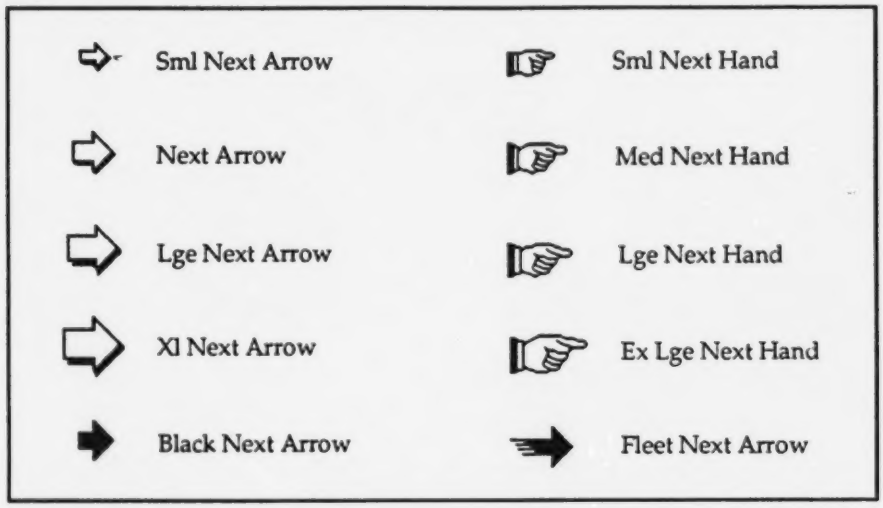
Yet another problem with multiple backgrounds is that they consume a lot of disk space. Cutting down on the number of backgrounds will reduce the size of your stack.

On the other hand, you should make use of the power of backgrounds. If you have a button or field that needs to appear on every card, put it on the background. If there are just a couple of cards that don't use that particular button or field, you can cover the background objects with Transparent (invisible) buttons when needed. Think about redesigning your background if you find yourself putting the same object on card after card.

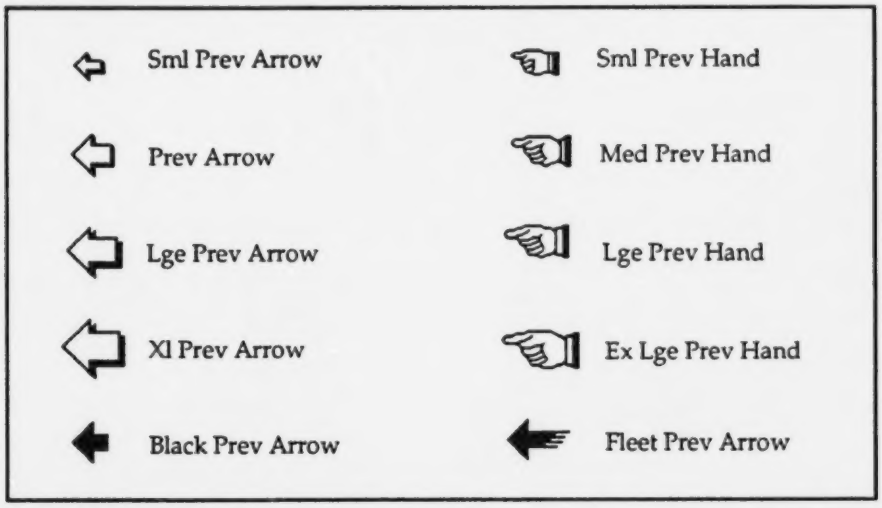
Finally, designing your background is as important as designing the structure of your stack. Carefully thinking about what should be on the backgrounds in your stack in advance of developing your stack will save you a lot of time at the computer keyboard.

Conventions for Using Icons

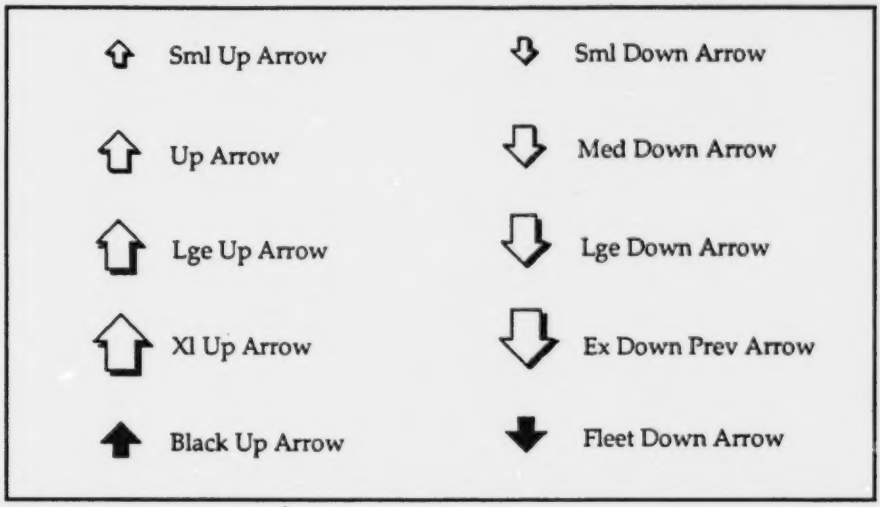
There are some conventions for using various icons on buttons. If you deviate too much from these conventions, users who have used other *HyperCard* stacks may have a great deal of difficulty getting around in your stack. For example, the following icons standardly mean "go to the next card" in the stack.



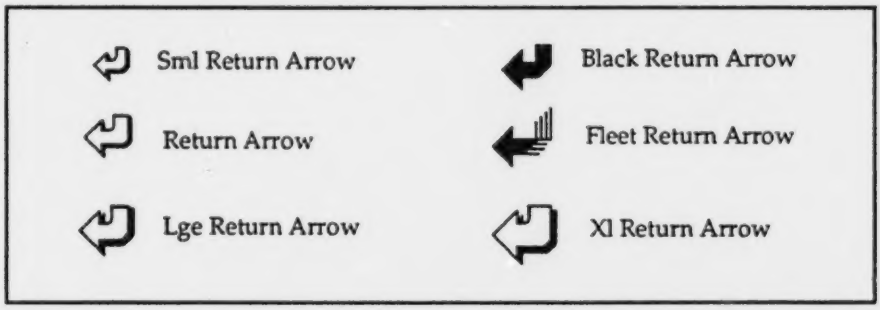
Similarly, these next icons usually mean "go to the previous card."



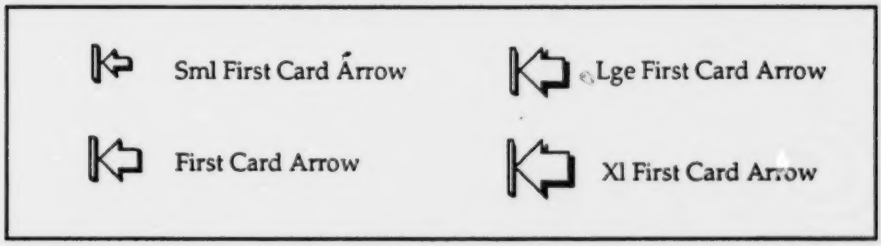
In addition, the following "up" and "down" arrows generally mean "go to the previous card" and "go to the next card," respectively.



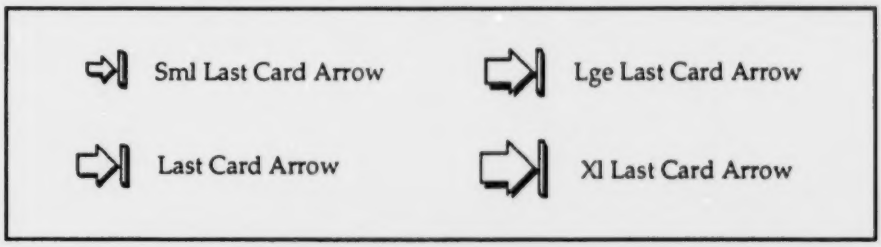
The following icons commonly mean "return." Return can have a lot of different meanings in *HyperCard*. Return can mean "go back to the menu," "return to the last branch in the stack," and so forth.



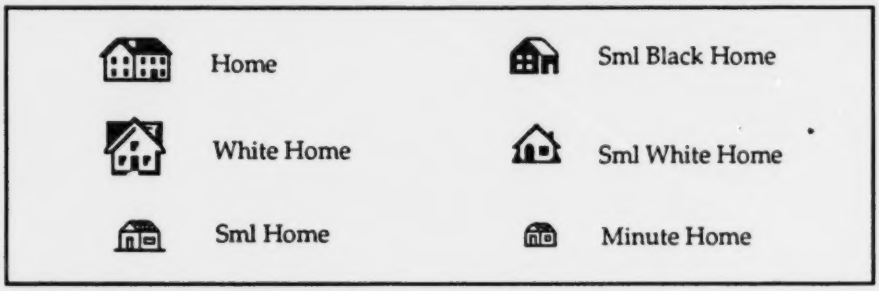
The following icons generally mean "go to the first card in the stack."



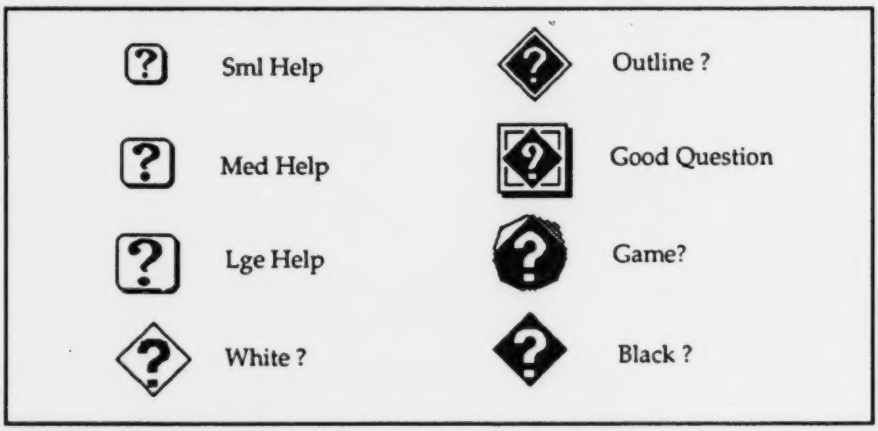
These next icons generally mean "go to the last card in the stack."



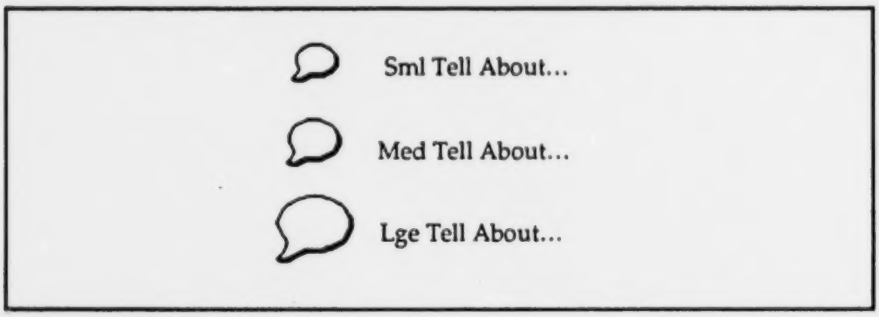
An icon that looks like a small house almost always means "go to the Home Card."



A question mark icon generally means that the user should click on the question mark for help.



Little "balloons" generally mean "tell me about."



Some Other Considerations

Every stack should be designed so that the user can "recover" from getting hopelessly lost. For example, if the stack is designed to be opened from the Home Card, it is a good idea to include a Home button on every card in your stack, or at least on cards that are easily reached at any time when using your stack. If the stack is designed as a stand-alone stack with

a clear beginning point (such as a main menu), every card should probably contain a button that returns the user to that card.

Another important consideration when designing stacks is consistency. If buttons and fields are not placed carefully, the transitions between cards will be unpleasant. Buttons and fields may appear to jump around if they are not in exactly the same spot on each card. You can avoid that problem by using the copy-and-paste features in *HyperCard*. A button copied from one card will automatically be pasted in exactly the same spot on another card.

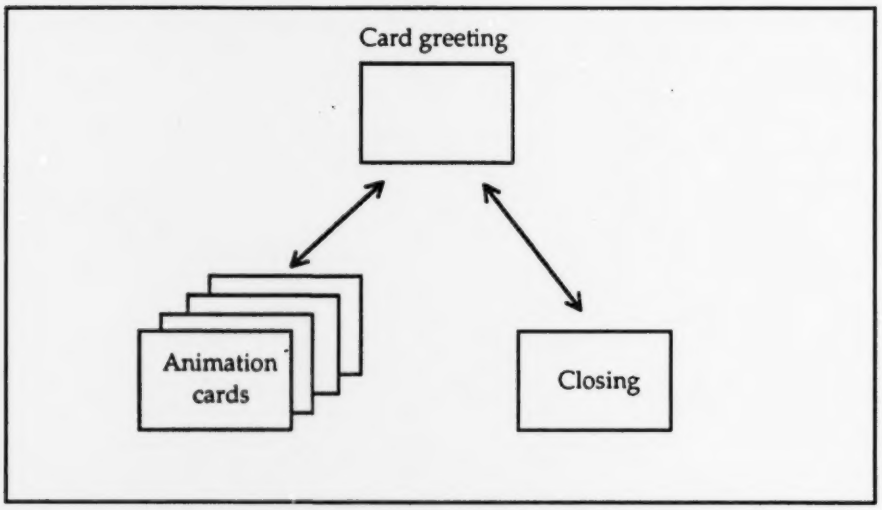
Even worse than having buttons appear to shift slightly as you move through the stack is having buttons that accomplish the same task appear at different places on every card. This forces users to search for buttons they need on each card.

If you use icons on buttons, make sure that the meaning of the icon is clear. Even better, put both the icon and a word on buttons to make their functions clear.

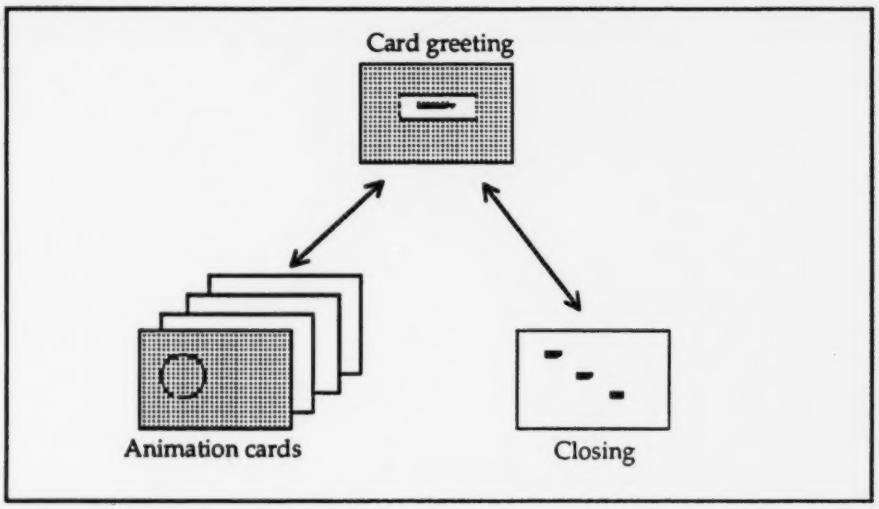
Consistency and clarity in placement and use of the various elements of your stack are quite important. Keep in mind that *you* are very familiar with the design of your stack. A new user is not. You must be sure that everything on your stack is easy to understand.

Tiny Cards—An Aid to Creating a Stack Map

It is easy to lose track of the structure of your stack as you are working, even if you planned it carefully in advance. To help visualize your stack, it is a good idea to make a stack map. A stack map looks like the diagrams at the beginning of this chapter, but typically the individual cards are identified. For example, you might have



HyperCard provides a method to help you make stack maps so that they can easily become a part of your stack and perhaps serve as a guide for the users of your stack. You can make "tiny cards"—miniature copies of the actual cards in your stack. The preceding stack map might be made to look like this:



To make these small-card images:

1. Select **Copy Card** from the **Edit** menu.
2. Move to the card on which you want your map.
3. Press ⌘-Shift-V (Command-Shift-V).
4. Move the image into place.

You can then use the Paint tools to draw arrows or add other details.

Activities

1. Select a stack you have created. Use the ideas from the Creating a Stack Map section of this chapter to create a stack map for your stack.
2. Select one of the "best" stacks you have created. Critique its design, paying special attention to the design considerations discussed in this chapter. Then redo the stack so that it is better designed.
3. Select a poorly designed stack that is intended to communicate a particular message. Discuss how the poor quality of the design decreases the effectiveness of the communication.
4. Get together with two or more other people and select a stack to be critiqued. Work together, noting where there is agreement and where there is disagreement. Discuss the idea of objective versus subjective criteria for judging the quality of a stack design.

5. One of the most important ideas in problem solving is building on previous work that you and others have done. Is a good stack design something that can be used by others? Discuss the idea of how one might store a collection of stack design templates in a way that would help other *HyperCard* users find a design to fit their particular needs.

Chapter 10. Scripting Beeps and Music

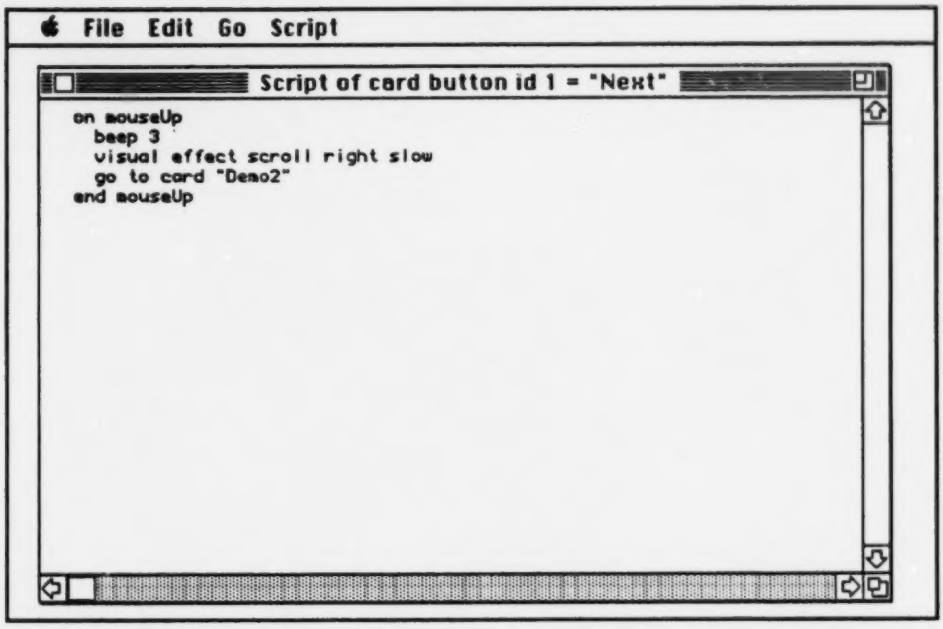
The *HyperCard* system is capable of producing two kinds of sounds. The first sound is a "beep"—the same beep used as an audio alert in a number of different Macintosh applications. The second sound is music. This chapter discusses how to script beeps and music. Beeps and music add a new dimension to your communication techniques.

This chapter also introduces the Message box, an important feature in the *HyperCard* system. You will find it useful in your explorations of music.

The Beep Command

Suppose you want an audible "beep" to be produced when a user clicks on a button. To do this, add the code **beep** to the button script. If you want more than one beep, include an integer after the word **beep**, telling the computer how many times the beep should occur.

The following illustration shows a button script that will produce three beeps before doing the visual effect as *HyperCard* moves to card Demo2.



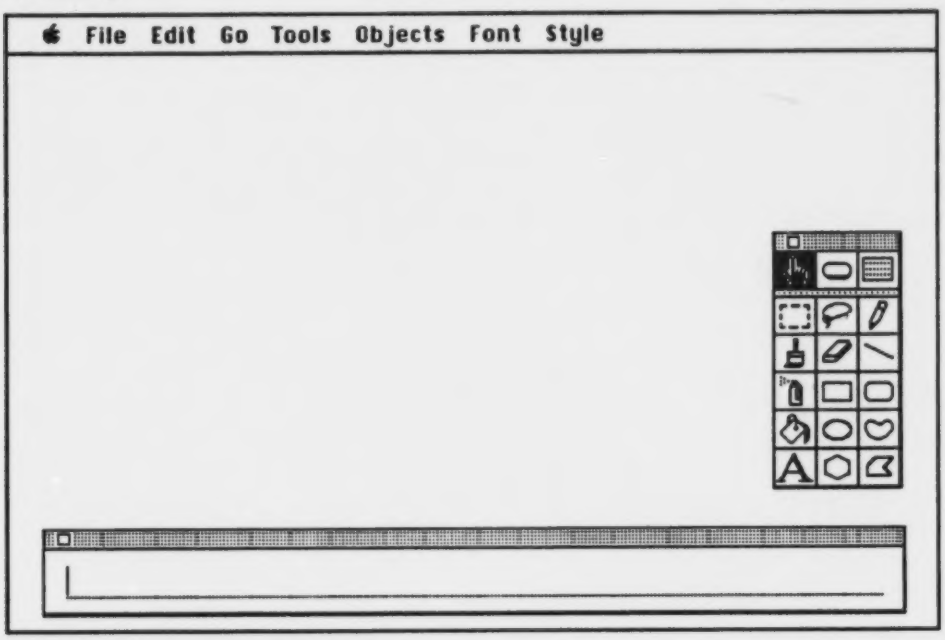
It is also possible to have *HyperCard* produce the three beeps *after* moving to the next card. That is accomplished by using the following button script:

```
on mouseUp
```

```
    visual effect scroll right slow
    go to card "Demo2"
    beep 3
end mouseUp
```

The Message Box

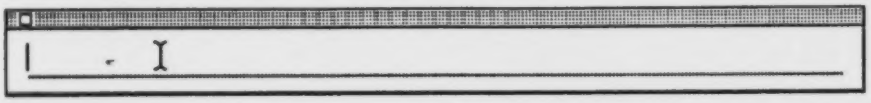
As illustrated above and in Chapters 6 and 7, it is easy to modify and add to a HyperTalk button script. It is also possible to use HyperTalk commands without writing a script. This is done by using the Message box. If you pull down the **Go** menu, near the bottom of the menu you see **Message**. If you select **Message** or press ⌘-M (Command-M), you see the following:



The Message box appears at the bottom of your current card with a vertical bar pointer (|) at the left side.

Technical Note: If you are using a screen larger than the "standard" 9-inch Macintosh screen, the Message box may appear below your card. Click in the gray area at the top of the Message box and drag it onto the card in the location shown in the previous illustration.

Select the **Browse** tool and position the hand-shaped pointer somewhere above the Message box. Then slowly drag the mouse into the center of the Message box. You see the pointer change from the pointing hand to an arrow to an I-beam. Whenever the pointer is over the center of the Message box, you will see the I-beam. You do not have to be using the **Browse** tool; no matter what tool you have selected, the I-beam appears when you are over the area in the Message box where you can type.



You can type HyperTalk commands in the Message box. For example, place the mouse pointer over the Message box and type

```
go to next card
```

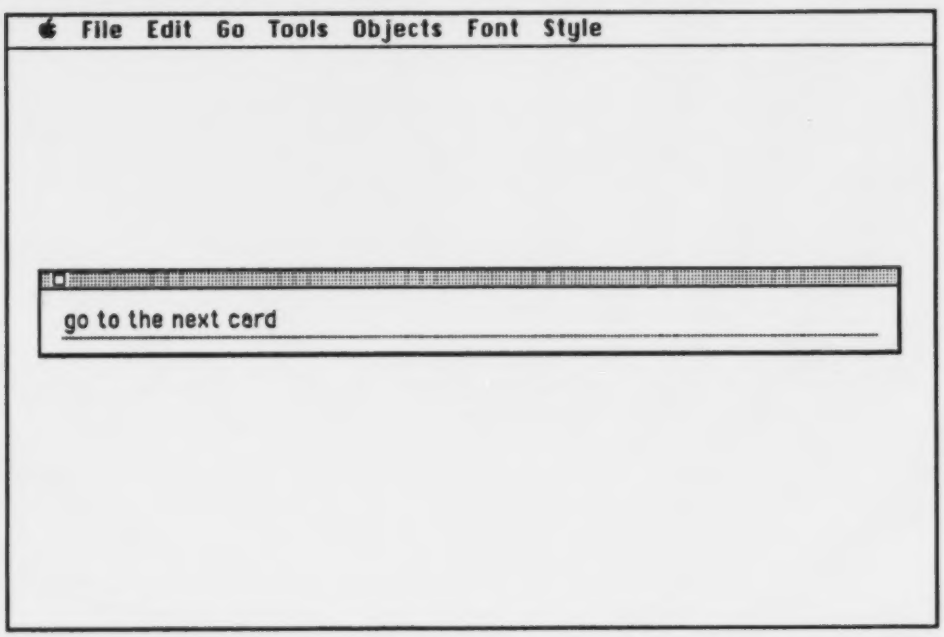
When you press the Return key, you are on the next card. The word "next" can always be used to refer to the next card in your stack. Note that if you have only one card in your stack, this command will have no effect.

When you move to the next card, the Message box appears in the same place on your destination card. Use the **Go** menu or ⌘-3 (Command-3) to move through your stack. The Message box is at the same place on every card in your stack. Thus, suppose that the Message box contains the

```
go to next card
```

HyperTalk command. If the I-beam pointer is in the Message box, repeatedly pressing the Return key will move you from card to card, allowing you to move quickly through the cards in your stack.

You can place the Message box anywhere on your screen. Move the mouse pointer over the shaded bar at the top of the Message box. Drag it to the location of your choice by holding down the mouse button and releasing it when the Message box is where you want it.



You can change the contents of the Message box by using the mouse and keyboard to edit what is already there. For example, you might

replace "next" with "previous" to go to the previous card. You can double click on "next"

and simply type "previous" or "prev" in its place.

If the vertical bar is flashing in the Message box, any text you type will be inserted after the location of the pointer. If you want to completely replace the text in the Message box, you can select it all and press the Delete key. You can also move the mouse pointer outside the Message box and click anywhere. When you begin to type, the **Browse** tool disappears and the text you type appears in the Message box, leaving the flashing vertical bar to the right of the text.

What do you think will happen if you type

```
beep 3
```

in the Message box and press the Return key? You should hear three beeps.

The Play Command

The **play** command allows you to play musical scales with different types of sounds. You can type **play** commands into the Message box and into button scripts. When you are first learning to use the **play** command, it tends to be faster and more convenient to use the Message box for musical experimentation.

HyperCard includes the sounds "Boing" and "Harpsichord." (If you are using *HyperCard* 2.0 or higher, you also have the "Flute" sound.)

Technical Note: You can also use digitized sound and voice samples that are part of the resources in your stack. Check a technical reference source, such as Inside Macintosh, Volume V, *for more information on sound resources.*

Try typing

```
play "boing" "c d e"
```

in the Message box and press the Return key. You hear three "boings" going up the scale. Next, try

```
play "boing" tempo 100 "c d e"
```

You hear the same three notes, perhaps at a different speed. Now change the 100 to 1000:

```
play "boing" tempo 1000 "c d e"
```

Now you hear the notes at a much faster speed. You can now see that the **play** command has three parts. The structure of this command is:

```
play "string1" [tempo number] "string2"
```

The first input to **play**, "string1," is the name of the sound. This may be one of the built-in sounds, or it may be a digitized sound. To avoid confusion, you should always put quotations marks around the name of the sound. Using "tempo" is optional; if "tempo" is omitted, *HyperCard* defaults to 120 unless tempo has been used in a previous **play** command. If it has been used previously, that value of tempo is used. "Tempo" has one parameter—a number representing the speed at which the sounds play. "String2," the third parameter, is the list of notes to play. It must contain fewer than 256 characters. Note that it is a good idea to put quotation marks around this list as well.

Representing Notes

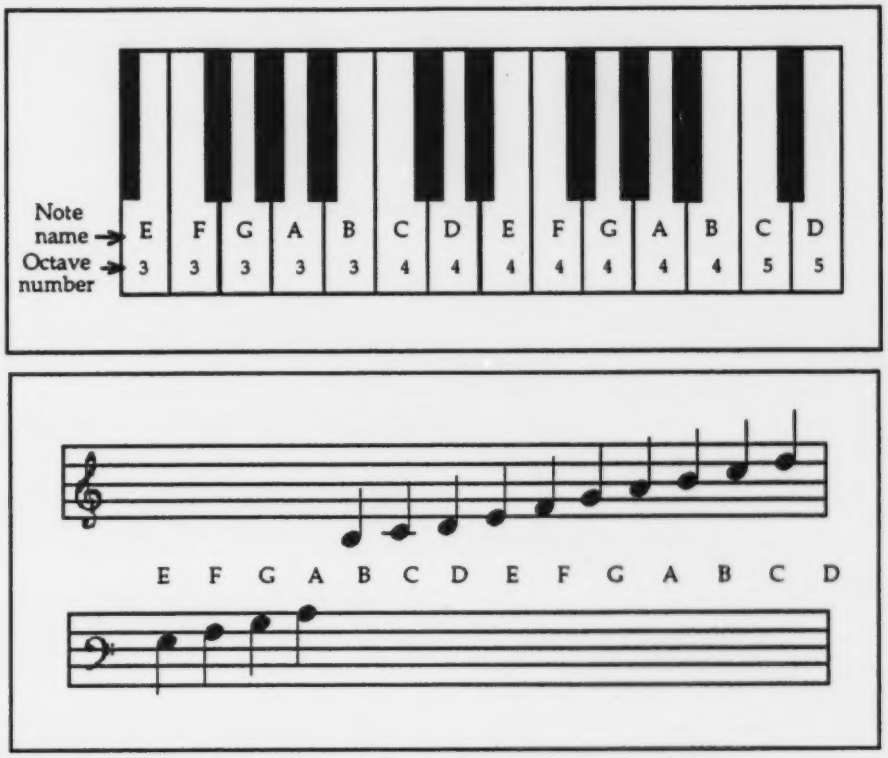
To begin exploring the **play** command, type in the Message box

```
play "harpsichord" "c4q c#4q d4q d#4q e4q f4q f#4q
   g4q"
```

and press the Return key when you have finished typing the line. You hear a scale. "C4q" represents a single note. "D#4q" represents another note. There are four possible parts to each note representation. First comes the note name. Notes are named using the conventional "A B C D E F G" notation. An "R" or a zero is used to represent a rest.

After the note name comes the *accidental*. An accidental is either a sharp (represented by #) or a flat (represented by b). Otherwise the note is natural (represented by nothing.) Thus, C#4q represents the note C sharp, while Cb4q represents the note C flat. C4q represents C natural.

Next comes the octave number. Higher numbers give higher pitches. The number "2" is probably the smallest number you will want to use to get musical sounds. If you do not include an octave number, *HyperCard* defaults to octave 4, the octave containing middle C.



Seven is probably the highest octave number you will want to use. If you use higher numbers, the pitches you hear will not be what you expect.

The final part of a note name is the *duration*. Duration is represented by

w whole note

h half note

q quarter note

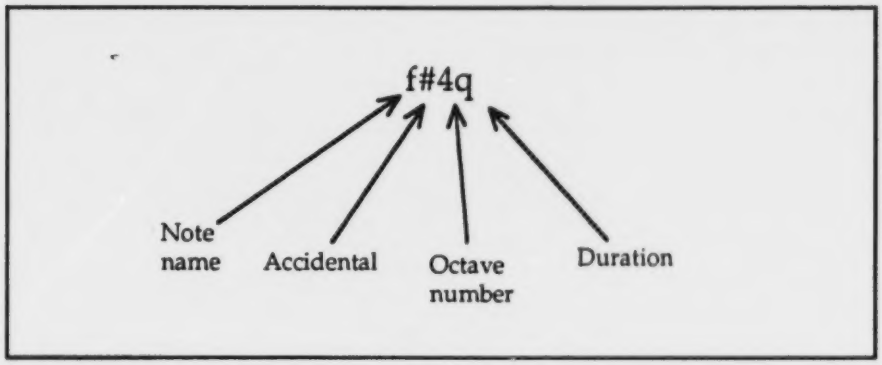
e eighth note

s sixteenth note

t thirty-second note

x sixty-fourth note

In addition, you can use a period after the duration to specify a dotted note (1 1/2 times the normal duration) or put the digit 3 after the duration to represent a triplet note (2/3 the normal duration). The longest possible note duration is slightly more than 16 seconds.



Note that if you do not specify a duration, *HyperCard* defaults to a quarter note.

Using the Play Command

There are many possible variations when using the **play** command. If you carefully examine the syntax of **play**, you notice that the tempo is optional. In addition, you do not have to specify the octave number and the duration for each note. Look again at the example given at the beginning of this chapter. Each note had the octave and duration specified. Once you specify an octave, the notes that follow will stay in the same octave until you change it. Thus,

```
play "harpsichord" "c4q c#4q d4q d#4q e4q f4q f#4q
   g4q"
```

can be written

```
play "harpsichord" "c4q c#q dq d#q eq fq f#q gq"
```

because all of the notes are in the fourth octave, the octave containing middle C. Changing the octave to 5 midway in the list will make all the notes beginning with the change play in the next higher octave. For example, if you type

```
play "harpsichord" "c4q c#q dq d#5q eq fq f# gq"
```

d# and all of the notes after it will play in the fifth octave. Duration also can be carried to subsequent notes. Thus, you can write

```
play "harpsichord" "c4q c# d d# e f f# g"
```

and hear only quarter notes. If you change the d# to d#h,

```
play "harpsichord" "c4q c# d d#h e f f# g"
```

all of the notes beginning with d# will be played as half notes.

The duration given with each note or group of notes tells *HyperCard* how fast to play that note or group of notes. When you include tempo in

the **play** command, you change the speed at which all of the notes are played. You saw earlier that if you type

```
play "harpsichord" tempo 100 "c4q c# d d# e f f# g"
```

you hear a slow scale, while if you type

```
play "harpsichord" tempo 1000 "c4q c# d d# e f f# g"
```

you hear a very rapid scale. Recall that if no value is given for tempo, *HyperCard* uses whatever value was last used with tempo.

A Button With Music

Suppose you create a button with music in the script, for example

```
on mouseUp
   play "harpsichord" tempo 100 "c4q c# d d# e f f# g"
   go to next card
end mouseUp
```

If you click on this button, the music continues to play even after the next card is visible. Music continues to play while other actions occur in *HyperCard*. You can control what happens with music by using

```
play stop
```

Modify the previous script as follows:

```
on mouseUp
   play "harpsichord" tempo 100 "c4q c# d d# e f f# g"
   visual effect dissolve slowly
   go to next card
   play stop
   go to previous card
end mouseUp
```

You will probably hear the music stop in the middle of the scale. After *HyperCard* encounters the **play stop** command, the music stops.

Technical Note: The number of notes you hear depends on the speed of the Macintosh you are using. The speed also depends on whether you are using a floppy disk drive or a hard drive. If you hear all of the notes, remove the "slowly" from the script.

Of course, the **play** command can be used to add little tunes to your stack or to play recorded sounds. However, you can also use **play** to add auditory prompts and cues to your stack. For example, you might put the following script on a button:

```
on mouseUp
   play "harpsichord" tempo 100 "g6s cs"
   go to next card
end mouseUp
```

129

This script creates a gentle, high-pitched sound when you click on the button. Such an auditory cue makes it clear to users that they have actually clicked on the button.

It is not uncommon for *HyperCard* users to decide that they can't work with sounds because they are not "musical." You don't have to know much about music to use the **play** command. If you borrow some beginning band music in the key of C and use the charts at the beginning of this chapter, you should be able to create *HyperCard* music with relative ease.

Making Mistakes While Using the Play Command

It is easy to make mistakes when using the **play** command. If you put a name of a sound that does not exist after **play**, you will hear nothing at all. You also need to be sure that the sound on your Macintosh is not turned off. If the sound is off, the Menu bar flashes when you would normally hear a "beep."

Be particularly careful about typing **play** commands. A misspelling of the sound name or a typographical error in some other part of the command can have unexpected results. Sometimes your music won't play correctly. Sometimes you will hear extra notes, or no notes at all. Similar problems can occur if you don't use quotation marks around the sound name and around the list of notes.

Note names can also be represented as numbers. Middle C is 60; the B below middle C is 59; the D above middle C is 61. Using numbers can be helpful in your future work with *HyperCard*, and this is one of the reasons it is important to be particularly careful about spacing and punctuation in **play** commands.

Bugs and Debugging

Directly writing and editing scripts in HyperTalk adds a new dimension to detecting and correcting bugs in a stack. As you work with HyperTalk, you will find that there are two major categories of bugs you will encounter over and over again:

1. Errors the computer can detect. These are errors that cause the computer to produce an error message. Often such an error results from an error in keyboarding, an error in the spacing in an instruction, an error in the punctuation or in the spelling of words in an instruction, or a misunderstanding of the details of how a specific instruction must be written. Such errors in the "grammar" of *HyperCard* are usually called *syntax errors*.

2. Errors the computer cannot detect. These errors do not result in an error message, but they produce an incorrect result. It is easy to forget to link a button or to link it to the wrong card. It is easy to keyboard go to card id "DEMO7" when you actually meant to keyboard go to card id "DEMO6." The computer has no way of knowing what you had in mind. We will call these errors in meaning *logic errors.*

Sound as Part of the Stack Design

Using sound in your stack can be a lot of fun; however, it is important to think about how the sounds will affect your users. If you put a song on a button that starts the user through your stack, that song may be impressive and a lot of fun the *first* time the stack is used. But after a few times, the song becomes a nuisance. This might be avoided by having a "next card" button with a script such as

```
on mouseUp
   play stop
   go next card
end mouseUp
```

The line "play stop" causes the sound to stop as soon as the user clicks the mouse on the "next card" button.

You may use a clever sound to make a transition from card to card or from section to section in your stack. That sound may be neat—and even appropriate—but users can quickly become exasperated by having to listen to the sound every time they make a transition from one place to another.

Listen to the sounds in your stack carefully. Think about your audience. Imagine what it would be like to use your stack over and over again. If *you* get tired of hearing a sound as you are developing your stack, it is likely that users who examine your stack more than one time will also get tired of the sound.

Activities

Even if you think you are not "musical," try writing some *HyperCard* music anyway! All you need to do is get some beginning instrumental music from the music department in your school or from a music store. If you are unfamiliar with traditional musical notation, you should get music in the key of C. This will avoid problems with those "accidentals" mentioned earlier in this chapter. Use the keyboard chart given in this chapter to help you get started.

1. Here is a note list for the first part of "Three Blind Mice":

```
eq4 d ch eq d ch
```

Create several buttons that will play this song fast, slow, in a high range, in a low range, and so on.

2. If you have musical experience, create a card with a series of buttons for musical exploration. Each could play a different tune, a different phrase of a longer work, or even a different interval. Use your imagination to create a card (or stack) that would help teach the user something about music.

3. You can create interesting sound effects by playing notes in very low or very high ranges or by playing them very quickly. Experiment with some groups of notes and create a series of sound effects buttons.

4. The scroll left and scroll right visual effects give visual images of moving to a previous or next card, respectively. Develop an audio effect that appropriately conveys the impression of moving to a previous or next card.

5. There are lots of ways to store sound, such as on a phonograph record, CD, or tape, or in a computer. Discuss the relative advantages and disadvantages of each medium in terms of the idea of building on the previous work of other people when solving problems.

6. Try making some deliberate errors in scripting. Develop a short list of the types of error messages that the *HyperCard* system generates when it encounters the types of errors it can detect.

Chapter 11. What Else Is There in *HyperCard?*

HyperTalk is a computer programming language that contains a wide range of features. A number of major features have not yet been discussed in this book.

This chapter contains a brief introduction to three major topics:

1. The overall nature of computer programming languages and their capabilities.
2. Representing problems so that they can be solved by computer.
3. Some features of HyperTalk that are useful in animation.

Technical Note: This chapter contains very brief introductions to very broad topics. There is no intent that the reader master these topics from the brief introductions. Rather, the intent is that the reader become aware that there is still much to be learned.

Programming Languages

Here are three very important facts about computers:

1. A computer can rapidly and accurately carry out a detailed, step-by-step set of instructions written in a language the computer is designed to "understand." This step-by-step set of instructions is called a *computer program.*
2. A computer can store a large amount of data. It can access, manipulate, and add to this data as it follows the instructions in a computer program.
3. People have developed a number of computer programming languages that are designed to be independent of the specific machine language of a particular make or model of computer. Software exists to translate programs written in these "higher level" languages into code that can be understood by specific makes or models of computers. (Because of this, HyperTalk is available on the full range of Macintosh computers.)

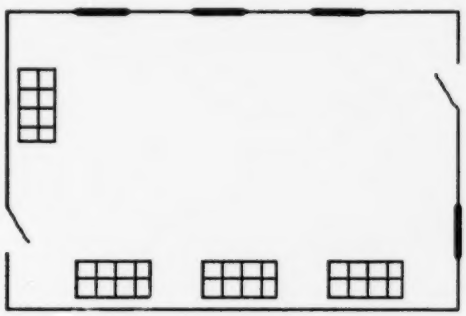
There are hundreds of different higher level computer programming languages. Often a particular higher level programming language is specifically designed to be domain specific—for example, to fit the needs of business people or to fit the needs of mathematicians. HyperTalk was

designed for people who want to develop nonlinear multimedia communications.

Computer programming can be thought of as a type of problem-solving activity in which one uses a programming language to represent a problem and its associated data, and to specify a set of directions on how to solve the problem. The next section of this chapter contains a brief discussion of problem and data representation.

Modeling (Representing) Problems

Consider the graphic of the room layout you have seen previously in this book.



This was created using the graphics tools in *HyperCard* and is thus easily included in a *HyperCard* stack. The *HyperCard* system is specifically designed to make it easy to create and store such graphics. Such graphics might be particularly useful in solving a room-layout problem.

The room diagram is not an actual room—it is a model or graphic representation of a room. There are lots of different ways to model or represent a problem. Different representations will prove to be better and/or more appropriate for solving different problems. Here is a brief list of some commonly used methods for representing a problem.

Mental Models

When you think about a problem, you are using a mental representation of the problem. Mental models are particularly useful because they are easily changed. That is, the mind is both a storage and a processing device, and it has great versatility.

Written Models

The development of reading and writing is clearly one of the most important intellectual achievements of the human race. Reading and writing are useful both for storing and processing information. With reading and writing, one can accumulate knowledge over a long period of

time, share it among a number of different people, and pass the knowledge on to future generations.

Imagine a written description of a room, complete with diagrams, drawings, blue prints, and so forth. This would be a written symbolic model of the room.

Scale Models

You have seen scale models of cars, airplanes, and houses. A scale model of an office building can be used to help design the office building. What will the office building look like when it is built? How will it fit into the landscape? Will people find the office building attractive?

Airplane designers build scale models of airplanes. A scale model of an airplane can be used in a wind tunnel to test the shape of the body and wings. How much air resistance will there be? This is much cheaper than building an entire airplane and then seeing how well it flies. It is also much safer!

Mathematical Modeling

You may have wondered why you have to take so much math in school. The reason is that math is a general-purpose aid to modeling many different kinds of problems. For example, how much money do you have with you right now? The answer—a number—is a mathematical model for your money.

Think about a carpenter building a table. The table will have a rectangular top with a molding around it. The carpenter must think about how much material will be needed. Here is a mathematical model for this problem.

A = LW	Area = Length x Width
P = 2L+2W	Perimeter = 2 x Length + 2 x Width

Computer Models

The computer was invented as an aid to mathematical modeling. The idea was that certain types of math formulas could be stored in a computer, and the computer could automatically do the work of using the formulas. The goal was to make it much easier to solve problems that require a great deal of computation.

Soon after the first computers were built, people began to think of lots of uses for computers outside of math. For example, a computer could store a written symbolic model. A computer could be used in place of scale models in many architectural situations. Computer models combine

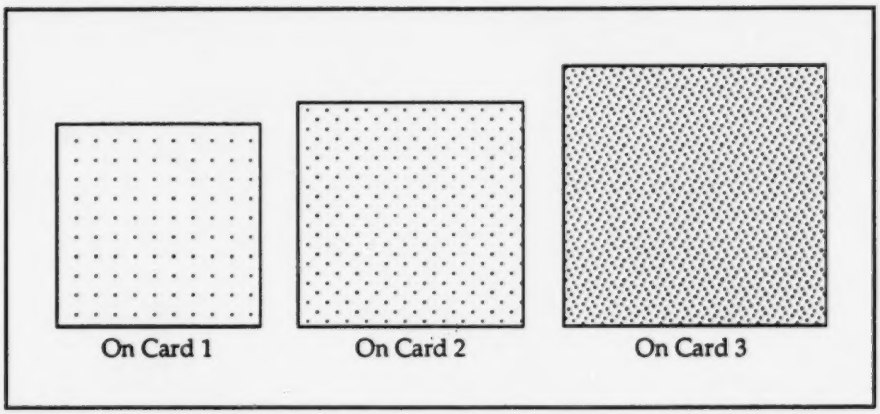
some of the best features of mental models, written symbolic models, scale models, and mathematical models. Thus, computer models are now in wide use.

In solving a problem by computer, you are apt to use several different representations. You may begin with a mental model, draw a sketch and/or write a brief description of the problem, and then begin to develop a computer model. Through training and experience you will gradually develop an intuitive feel for how much time and effort to put into planning and thinking before actually beginning to work at the computer.

Animation by Flipping Cards

One simple way to create animation is to create a series of cards, each with a slightly different picture on it. Then you can flip through the cards to produce an animation effect, much like you might flip through a real deck of cards or flip book on which each card or page has a slightly different picture.

To illustrate this, begin by creating a card with a filled square on it. Then make a slightly darker and larger square on the next card and a yet darker and larger square on the third card. Draw these so that the bottom left corner of each square is at the same location on each card.



If you now "flip through" the cards, you will see the square get larger and darker. You can flip through the cards by using ⌘-3 (Command-3) to go quickly to the next card.

When you are satisfied with your animation, add an "Animate" button to your first card. There is a good shortcut for creating buttons. Choose the Button tool, then hold down the Command key. Click and drag, and a transparent button appears. (The same technique works for fields.) Make the button any style you want and then add a script to flip through the cards. You might write

```
on mouseUp
   wait 15 ticks
   go to next card
   wait 15 ticks
   go to next card
end mouseUp
```

A tick is 1/60 of a second. This script "flips" to the second card and then to the third card. The actual time it takes to flip depends on the speed of the computer you are using. Thus, you may need to change the number of ticks of waiting time to achieve the desired effect.

This particular animation technique works fine for a few cards. However, if you have a lot of cards to animate, the number of lines in your script can get quite long. There is a more efficient way to repeat actions a number of times. You can use the **repeat** keyword.

The script on the Animate button could be written

```
on mouseUp
   repeat 2 times
      wait 15 ticks
      go to next card
   end repeat
end mouseUp
```

To use **repeat,** you need to know four things.

1. **Repeat** must be followed by a number (on the same line) that indicates how many times to repeat the action.
2. The word "times" follows the number on the same line.
3. The action to be repeated follows; it may be one line or several lines long.
4. The repeated action is followed by the words "end repeat" on the next line.

Interactive Programming

No doubt you have used many interactive computer programs that asked you questions and requested a response from you. The computer then uses your response, for example, to make a decision or respond back to you. *HyperCard* contains a variety of provisions to do interactive programming. One of these is called **answer**.

Here is a simple example. If you type in the Message box

```
answer "Go for it!"
```

you see

For a use of this capability of HyperTalk, suppose that a card contains a detailed set of directions to the user. The "Next Card" button could contain the script

```
on mouseUp
   answer "Go for it!"
   go next card
end mouseUp
```

When the user clicks on the "Next Card" button and then clicks on OK or presses the Return key, the program would then move to the next card in the stack.

The general format for **answer** is:

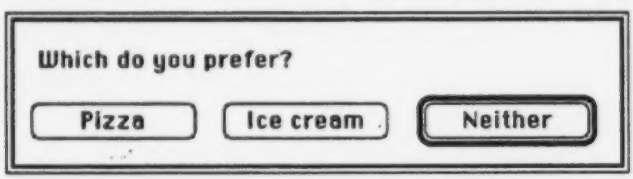
answer *string1* [with *string2* [or *string3*] or *string4*]]]

"String1" is the question to be asked; "string2," "string3," and "string4" are the possible replies. If you look carefully at the format of **answer**, you see that you can have up to three responses. For example,

```
answer "Which do you like?" with "pizza" or ¬
   "ice cream" or "neither"
```

Technical Note: For ease of readability, it is sometimes desirable to break a line of code. Hold down the Option key and press Return and it will produce the symbol ¬. Do not break in the middle of a string, for example, between "ice" and "cream" in the quoted string "ice cream."

If you type the above **answer** command into the Message box, you see



Note that the button on the far right is always highlighted. The highlighted button is the one selected if the user presses the Return key. Also note that "question" and "reply" are limited in length by the size of the dialog box and the size of the buttons. That is to say, there is no "word wrap" allowed for the question and you cannot have multiline buttons.

When you click on a button in an **answer** dialog box, the word or words inside of the button are stored in a special location in the memory of the computer called "it." **It** is a predefined variable. That is, **it** may contain any of the possible responses to an **answer** command.

Because the value of the answer is stored, you can test that answer. Here is a script that uses the previous **answer** command and responds differently depending on which answer you choose. This script uses the **if** statement, another key feature in HyperTalk. The **if** keyword allows branching.

```
on mouseUp
   answer "Which do you like?" with "pizza" or ¬
      "ice cream" or "neither"
   if it = "pizza" then put "I like pizza too." ¬
      into Message box
   if it = "ice cream" then put "I LOVE Ice cream!" ¬
      into Message box
   if it = "neither" then put "I'll eat alone, then."¬
      into Message box
end mouseUp
```

The progress of the program depends on the results of the **if**. In the above **if** commands, **it** is tested against each answer. If the result of the test is true, the sentence after **then** is put into the Message box. If the result of the test is false, HyperTalk moves to the next statement in the script.

Programming Language Constructs

The scripts given in this chapter illustrate three major *constructs* used in computer programming. First, there is the construct of a linear sequence of computer instructions. The computer carries out the first instruction, then the second instruction, then the third instruction, and so on. Second, there is the construct of repetition. The computer repeatedly carries out a certain set of actions. Third, there is a construct of branching. The computer carries out one of several alternative actions. This is illustrated in the *HyperCard* script discussed above that makes use of **answer** and **if**.

The three constructs—sequencing, repetition, and branching—are the only constructs needed in a general-purpose computer programming language. That is, any problem that can be solved by a computer requires the use of only these three constructs. This means that after a programmer masters these three ideas, it is easy for the programmer to make High Road transfer of the ideas to any programming language.

The previous section also illustrated the use of a variable named **it**. Variables are an important part of a general-purpose programming language. They are used in representing complex problems and in expressing complex sets of directions.

Activities

1. Chapter 1 contained the following activity:

   Write in your journal about any previous *HyperCard* experiences you have had and summarize your current level of knowledge and skill. Think of this as a snapshot to be used in a "before" and "after" report on your progress in learning to use *HyperCard* effectively.

   Do the "after" part of the Chapter 1 activity and analyze the progress you have made.

2. Select a problem that interests you. Create mental, written symbolic, and *HyperCard* models for this problem. Discuss the relative value of each type of model for the particular problem you have selected.

3. Consider mental, written symbolic, scale, mathematical, and computer models. For each model, give an example of a type of problem where that model is "better" than the other four models.

4. Locate some clip art that interests you and create an animated story. The buttons could be used to cause a variety of actions to occur.

5. Create some animated "modern art." You could use a variety of geometric shapes and patterns to create a "painting."

6. Experiment with using **answer** and other features of HyperTalk illustrated in the section on interactive programming.

Appendix A. Keyboard Shortcuts

Navigation Between Cards	
Tilda (~/ `) key	go back
Command-1	go to the first card of current stack
Command-2	go to previous card
Command-3	go to next card
Command-4	go to the last card of current stack
Command-H	go to the Home stack
Command-R	show the most recent cards

Editing	
Command-X	cut
Command-C	copy
Command-V	paste
Command-Z	undo
Command-B	toggles edit background

Working With Objects	
Command-N	create a new card
Command-Option drag	copy a selected button or field
Command- +	bring an object closer
Command- -	send an object farther away
Command-drag	create a new button or field when the appropriate tool is selected
Command-P	print a card

Files	
Command-O	open a stack
Command-Q	quit *HyperCard*

Selecting Tools	
Command-Tab	Choose Browse tool
Command-Tab-Tab	Choose Button tool
Command-Tab-Tab-Tab	Choose Field tool

Show/Hide	
Command-space	show/hide Menu bar
Command-M	show/hide Message box
Command-Option	display buttons
Shift-Command-Option	display buttons and fields

Double Click on a Tool	
Select tool	selects the full screen
Lasso tool	selects all the graphics on the screen
Pencil tool	sets fatbits mode on/off
Brush tool	displays brush shapes window
Eraser tool	clears all graphics on the screen
Line tool	displays line sizes window
Rectangle	sets draw filled mode on/off
Round rectangle	sets draw filled mode on/off
Curve	sets draw filled mode on/off
Oval	sets draw filled mode on/off

Polygon tool	sets draw filled mode on/off (but will not draw filled)
Spray can	shows/hides patterns window
Bucket tool	shows/hides patterns window
Text tool	displays text styles window
Regular polygon	displays polygon sides window

Shift With Click and Drag	
Selected graphic	can move the graphic horizontally/vertically
Pencil tool	constrains to horizontal or vertical lines
Brush tool	constrains to horizontal or vertical strokes
Eraser tool	constrains to horizontal or vertical erasures
Line tool	constrains lines to 15-degree intervals
Spray tool	constrains to spraying horizontally or vertically

Command With Click and Drag	
Select tool	selects only a rectangle surrounding a graphic
Pencil tool	fatbits mode on/off
Lasso tool	select all
Brush tool	brush uses white pattern
Spray tool	spray uses white pattern

Option With Click and Drag	
On a selected graphic	copy that graphic
On Select tool	same as Lasso tool

Command and Shift With Click and Drag	
Spray tool	spray horizontally/vertically in white pattern

Shift and Option With Click and Drag	
Selected graphic	copies and moves horizontally/vertically

Command and Option With Click and Drag	
Pencil tool, FatBits mode	moves the window around

Delete Key	
Selected graphic	removes the graphic

Appendix B. Visual Effects

A button can be scripted with a visual effect. For example:

```
on mouseUp
   visual effect scroll left
   go to card "Demo2"
end mouseUp
```

Be sure you don't remove the "go to" statement. Visual effects occur only when you move from one card to another.

Here is the format of **visual**:

```
visual [effect] effectName [speed] [to image]
```

The following is a list of visual effects.

barn door close	barn door open	checkerboard
iris open	iris close	venetian blinds
zoom close	zoom open	dissolve
zoom in	zoom out	plain
scroll down	scroll up	
scroll left	scroll right	
wipe down	wipe up	
wipe left	wipe right	

If you are using *HyperCard* 2.0, you can also use the following *effectNames:*

shrink to top	shrink to bottom
shrink to center	stretch from top
stretch from bottom	stretch from center
cut	

In the syntax for **visual**, there are two optional pieces of information. *Speed* is the first optional parameter and can be one of the following:

```
fast
very fast
slow[ly]
very slow[ly]
```

The next phrase–[to *image*]–is also optional. While the entire *phrase* is optional, the "to" *must* be used if you use a value for "image." *Image* can be one of these terms:

```
black
gray (or grey)
inverse
white
card
```

146

Appendix C. Research on Problem Solving

There has been substantial research on problem solving. This research is strongly supportive of the idea that people can get better at problem solving by study and practice; it also supports the idea that the computer is a useful aid to problem solving.

This appendix contains a brief summary of some of the key research findings from the field of problem solving. It is based on work done by Billy C. Yates and Dave Moursund (see References) and focuses on 16 statements about problem solving that are strongly supported by the research literature. One can probably find counter arguments or contrary positions to each of these statements. However, the evidence to support each of these assertions is strong enough to convince many educators and learners that it is appropriate to consider these ideas when making educational and learning decisions.

1. To become a "world-class" expert in a particular area requires both considerable natural talent and at least 10 years of hard work.

Many world-class experts in such diverse areas as music, tennis, and chess have been studied. Two key findings emerge. One is that it takes many years of hard work to achieve expertise. The second is that a key part of the expertise seems to be that of having committed to memory approximately 50,000 patterns and what to do when one encounters these patterns (or slight variations thereof).

The number 50,000 suggested here might better be represented by a range, such as 25,000 to 100,000. It is a large number, but well within the capabilities of many humans. For example, it is likely that you have a sight-reading vocabulary in excess of 100,000 words. You can check this by selecting a very comprehensive dictionary and running a little test on yourself. Determine the percentage of words you recognize from a few random pages and take that percentage of the number of words in the dictionary. A very comprehensive dictionary may have a half million words in it.

The educational implications of this seem clear. Expertise comes from a combination of talent and hard work. If we want to produce world-class experts in a field, we need to create a supportive environment that keeps students working hard for the requisite number of years.

Many educational leaders suggest that we should design an educational system that strongly encourages each student to gain a broad general education and to specialize in some area. They suggest that a

student should pick an area of specialization relatively early—in fact, as early as the beginning of secondary school—and work hard on it for many years.

2. Problem solvers who talk about the steps they take to solve a problem do better than those who do not describe their efforts.

The key here appears to be a conscious effort to verbalize the processes one is going through in attempting to solve a problem. Perhaps the verbalization provides the problem solver with oral feedback. Perhaps procedural steps become better defined when one verbalizes them. Perhaps such self-talk increases metacognition; various research studies support the value of placing increased emphasis on metacognition as a way of improving higher-order thinking skills.

It could be that writing out one's thought processes is similar to verbalizing them. The idea of "journaling" in a variety of courses, including mathematics and computer programming courses, has gained support in recent years. The idea is to have students write down their thoughts and feelings about the learning processes that are involved in the topic they are studying. Similarly, it is often argued in the computer field that students who have to develop computer programs to solve a particular problem come to "really understand" how to solve the problem.

3. How we think about (or represent) a problem is a better indicator of the problem's difficulty than any quality intrinsic to the logic of the problem.

In essence, this says that the difficulty of a problem depends on the problem solver and how the problem solver happens to represent the problem. A problem that is very difficult for one person to solve may be quite easy for another person to solve, even though the two individuals are quite comparable in overall ability and training. This suggests that students should be encouraged to explore a variety of representations for a problem. A person with a large repertoire of problem representations may do better at dealing with novel problem situations.

In the representation of problems, we find that computers are quite useful in some cases and not at all useful in others. For example, a computer can easily present data in a variety of graphical formats, including line graphs, bar graphs, and graphs of two- and three-dimensional mathematical functions. But a computer is no substitute for the "doodling" and similar types of graphical "memory-mapping" activities that many people use when attacking problems. Suppose that one's mental representation of a problem is done in terms of analogy and metaphor. Research into the workings of the minds of successful researchers and inventors suggests that this approach is common and

perhaps necessary. A computer may be of little use in manipulating this type of mental representation.

Professional computer programmers realize that the choice of the programming language can make a substantial difference in the difficulty in solving a particular problem. A particular problem may be almost trivial in one language and nearly impossible in another.

For example, consider a business-oriented problem that neatly fits into a spreadsheet setting. It probably would be very difficult to solve the problem by writing a program in Logo or in HyperTalk. Now consider a recursive geometric task that fits well into the Logo domain. It might be very difficult to accomplish in HyperTalk, and absolutely impossible to accomplish using spreadsheet software.

It is clear that the computer programmer who knows a variety of programming languages and application packages has a distinct advantage over the person who knows only a single programming language or application package.

4. Problem-solving skills used in groups do not necessarily transfer to individual problem-solving situations.

The effectiveness of cooperative learning and cooperative problem solving is supported by a substantial body of research literature. However, transfer theory suggests that if we want students to perform as individuals, we will have the best results if we specifically train them to function as individuals.

Thus, we are faced with the task of deciding how much educational effort we should put into helping students develop group problem-solving skills and how much effort we should place on developing individual problem-solving skills.

We can couple this educational issue with the issue of having students gain both breadth and depth in their education. A broad general education, with a common core for all students, is needed to facilitate communication among people. But depth of education is needed to solve hard problems. This suggests that we need to create cooperative learning environments where people with a common core of knowledge and skills but with widely varying depth of expertise come together to work on hard, interdisciplinary problems. This appears to be an excellent environment for practicing the group problem-solving skills needed in the real world.

5. Even with well-defined problems, people tend to formulate small subgoals and may not be able to explain why they did so.

This suggests that even a well-defined problem can easily overwhelm the typical problem solver. When placed in such a problem-solving

situation, people tend to focus on small pieces of the task, to take some action, to do something. This process of thrashing around may get various parts of the subconscious engaged and allow time for relevant ideas to emerge.

When a problem seems complex and overwhelming, the idea of systematically breaking a big problem into smaller pieces becomes very important. All people need to learn this basic approach to problem solving and to gain considerable skill in carrying out this task. Computer programming (structured programming, top-down analysis) focuses on this idea as a central theme. Thus, students in computer programming classes get substantial practice in top-down analysis. Some research evidence supports the transfer of this skill to problem solving in areas outside of computer science. Research on transfer theory suggests that if computer science instructors placed greater emphasis on helping students find examples from outside of computer science, more transfer would occur.

6. The degree of consciousness we have about our thinking processes while solving a problem depends on whether we are using a familiar strategy or developing the strategy as we work.

One model of how people solve problems is that in essence people look for patterns that seem familiar and then apply standard strategies that they have previously found useful when the patterns occur. If we are attacking a problem and fail to distinguish familiar patterns, we tend to fall into the behavior described in Statement 5. As we move out of the realm of familiarity, we are less conscious of the types of thinking processes we are applying.

It is difficult to even think of ways to attack a problem that seems entirely new to us. However, we can get better through practice. It is helpful to be repeatedly placed in unfamiliar problem-solving situations.

7. We have a few basic problem-solving strategies for dealing with a variety of problem situations.

The research suggests that each person develops his or her own preferred set of techniques or heuristics (rules of thumb) to apply to a problem. Not everybody approaches the same type of problem in the same way, and success can be achieved through a variety of approaches.

Not everybody has the same basic set of problem-solving strategies, and most people have a quite limited set of strategies. If we can help students acquire even a few additional strategies, we may be making a major contribution to their overall ability to attack a variety of problems.

Studies in metacognition strongly support the idea that students should become consciously aware of the problem-solving strategies they

are using when engaged in learning about and practicing problem solving. For example, we can teach students to ask, "Is this a problem others have faced? Can I get some help by talking to other people or looking for information in a library?" If the answer is "yes," using a computer information retrieval system might be one approach, while talking to others might be a quite different but equally productive approach.

As two other problem-solving strategies, consider the examples of drawing a picture and making up a similar but simpler problem and solving it. There are a large number of such strategies. Research suggests that problem-solving skills are improved by learning and practicing such strategies over a wide range of problems.

8. Precise thinking (precise mental processing) is one of the keys to good problem-solving ability.

Here we are referring to careful, logical, justified reasoning in moving from step to step in solving a problem or presenting an argument. It is a type of mental processing emphasized in scientific method, mathematical analysis, computer programming, debate courses, law courses, and other academic situations. The main point is that precise thinking is a necessary part of every academic discipline in which people have ideas and points of view that they want to communicate orally or in writing. Practicing such careful communication improves one's ability to solve problems in these disciplines.

9. Changing our perspective on a problem often aids in arriving at a solution.

Much of Edward de Bono's work (see References) has been aimed at helping people learn to change their perspective on a problem. Many other writers have developed materials that give students practice in looking at a problem in a variety of different ways. Brainstorming is a standard technique, and it can be done individually or in a group.

We also have often seen that people who are quite successful in one field and then switch fields are able to bring a fresh perspective that may help them make major contributions to their newly chosen field. This has led to the development of interdisciplinary fields, such as physical chemistry or molecular biology. Part of what may be going on here is that the methodology or modes of inquiry in a particular discipline may be so narrowly defined that they are not adequate for addressing some of the problems that arise in the discipline.

One way to think about this is in terms of the resources (rules of the game, restrictions, allowable types of operations) that are part of the definition of a problem. A person working in a particular field gradually internalizes a set of resources and restrictions that are standard in attacking problems in that field. However, a person coming into the field

from another discipline is not constrained by this set of resources and restrictions and may discover an innovative new way to solve a problem.

Studies on major scientific breakthroughs suggest that this may indeed be a common occurrence. Thus, while domain specificity is very important in problem solving, domain-specific thinking can keep one from drawing on one's full range of knowledge and perhaps from discovering innovative new approaches to solving a problem.

10. Experts do no better than novices when solving problems outside of their domain of expertise. Good problem-solving ability in one area does not automatically carry over to problems in another area.

This statement about transfer of learning is somewhat contrary to the ideas discussed in Statement 9. It says, for example, that a person who is a brilliant mathematician is not necessarily good at helping students learn (that is, at being a good teacher) or at handling the internal politics of running a mathematics department (that is, at being a good administrator).

To be an expert in an area requires a tremendous amount of domain-specific knowledge. (In Statement 1 we discussed evidence that supported the contention that a high level of expertise is acquired through a minimum of 10 years of hard work in a field and the memorization of many thousands of patterns.)

Some people who develop and teach courses on problem solving have taken a position that is somewhat contrary to Statement 10. In essence, they argue that there are domain-independent problem-solving techniques that people can learn and can learn to apply within any discipline. For example, one useful technique in problem solving involves building on the work of others. A person can become quite skilled at doing library research on what is known about a particular problem. These library research skills are specific to the domain of library science but are also domain-independent in the sense that they can be applied across a broad range of academic domains.

11. The computer is an excellent problem-solving aid for certain types of problems.

Humans were quite adept and successful at problem solving long before computers were invented. But computers are a unique new tool that bring entirely new capabilities to the problem solver. Indeed, computers can solve certain categories of problems that are difficult or nearly impossible to solve without this aid. In addition, using the huge data storage, quick retrieval, and fast-processing capabilities of computers, it is possible to easily solve a wide range of problems that are challenging to even the best-trained person who is not using a computer.

The range of problems that computers can solve or make a significant contribution to solving is steadily growing. The challenge to our educational system is evident. We want students to understand the capabilities and limitations of computers as an aid to problem solving within each domain that students study as well as outside of any specific domain. The idea of integrating the computer as a tool for problem solving into the everyday curriculum has received strong support from a variety of researchers.

12. Groups of two to four individuals are better than larger groups or individuals at problem-solving activities that use computers.

This is a statement about cooperative problem solving in a computer environment. It says that a group of two to four people can outperform a single individual and can outperform a larger group over some range of tasks that researchers have studied. Thus, it states that "two heads are better than one," but it also suggests that "too many cooks spoil the broth."

Note that many computer problems are so large that it is only feasible to attack them through the efforts of a large group. However, the most successful approach is one that segments the problem so that the pieces are of a manageable size for smaller groups or individuals.

13. A prodigious memory may not enhance problem-solving skills; in fact, a set of carefully refined problem-solving strategies can be a more significant influence on performance in a given domain of discourse.

This is closely related to Statement 1, and the educational implications of this seem clear. One approach to improving problem-solving skills within a domain is to commit to memory more and more patterns and the correct responses when these patterns are encountered. If one can be a world-class expert by recognizing 50,000 patterns, what might one be by recognizing 500,000 or 5,000,000 patterns? The answer seems to be that this additional level of memorization is only of marginal use.

The issue is one of memorization versus learning to think by using what one has memorized. On one end of the scale one can imagine having no factual knowledge in a domain—not knowing the vocabulary or notation and not having ever encountered a problem from the domain. On the other end of the scale, one can imagine having studied and memorized many millions of individual problems and their solutions. Between these two extremes, one can imagine having learned a great many or a very few general problem-solving techniques—how to think with and make use of one's memorized knowledge.

The goal is to optimize a student's ability to solve problems within the domain and to learn to solve new problems. Research suggests that the best option is to have a balance between rote memorization and learning

to think by using the facts one has memorized. It seems evident that the optimal balance varies among individual students and among domains. Thus, to a large extent each student needs to learn how he or she learns and must learn to take responsibility for developing his or her own strategies for mastering a field.

14. People who have a positive attitude toward a problem-solving task do better than those who have a negative attitude.

There have been many research studies on attitudes and how they affect learning, retention, and problem solving. Generally there is agreement that having a positive attitude is desirable and contributes to learning and the ability to solve problems. This is related to the ownership component of a problem.

15. Problem solving in computer programming doesn't easily transfer to problem solving in other disciplines.

This topic has been extensively researched. Initially, researchers were confident that they would be able to find a solid relationship between time spent in computer programming courses and overall increased ability to solve problems outside of the computer programming domain. When initial results failed to support this hypothesis, other better designed studies were conducted. Some partial results have been reported. In general, if one breaks down problem solving into a number of subskills, one finds that there is transfer in some of them from computer programming. But that isn't at all surprising. Some of the subskills are nearly identical to subskills being taught in computer programming.

The main point of Statement 15 is that the teaching of computer programming is not a panacea to improving problem-solving skills. Computer programming is an important discipline in its own right, and it could well be an appropriate topic to include in almost any student's curriculum. But doing so will not contribute substantially to the overall problem-solving skills of students.

16. Our short-term, active memory is quite limited in capacity and is estimated to hold 7 ± 2 "chunks."

The limited active memory capacity of humans has been extensively studied. We know, for example, that most people can look up a seven-digit telephone number, walk across a room to a telephone, and key in the number. But if telephone numbers were eight digits in length, many people would be unable to recall them without developing some memory aids. A typical memory aid is "chunking." For example, I know that the area code for Oregon is 503 and that the prefix for the University of Oregon is 346. Thus, when I am out of state, I can call an on-campus friend at the University of Oregon by using the two chunks "area code for

Oregon" and "prefix for the University of Oregon," along with a four-digit sequence of digits.

Computers add a new dimension to the idea of chunking. Suppose I am working on a very complex mathematics problem. I might recognize that one piece of the problem requires solving a system of simultaneous nonlinear equations, a second piece requires solving a system of differential equations, and a third piece requires finding the area under a particular curve. Each of these is a task that can be carried out by a computer. I don't need to know the details of how the computer accomplishes these tasks in order to make use of the computer. In essence, I can learn that computers can solve certain problems. If I can break my large problem into component problems of the sort that a computer can solve, I can then solve my large problem.

References

Apple Computer, Inc. (1986). *Inside Macintosh: Volume V*. New York: Addison-Wesley.

Apple Computer, Inc. (1989). *HyperCard stack design guidelines*. New York: Addison-Wesley.

Beyer, B. K. (1984, March). Improving thinking skills: Defining the problem. *Phi Delta Kappan*, pp. 486-490.

Beyer, B. K. (1984, April). Improving thinking skills: Practical approaches. *Phi Delta Kappan*, pp. 556-560.

de Bono, E. (1971). *New think*. New York: Avon Books.

de Bono, E. (1973). *Lateral thinking: Creativity step by step*. New York: Harper and Row.

Fredericksen, N. (1984). Implications of cognitive theory for instruction in problem solving. *Review of Educational Research, 54*, 363-407.

Gardner, H. (1983). *Frames of mind: The theory of multiple intelligences*. New York: Basic Books.

Gazzaniga, M. S. (1988). *Mind matters: How the mind and brain interact to create our conscious lives*. Boston: Houghton Mifflin.

Glatthorn, A. A., & Baron, J. J. (1985). Good thinking vs. poor thinking. In Arthur Costa, *Developing minds: A resource book for teaching thinking*. ASCD.

Minsky, M. (1986). *The society of mind*. New York: Simon and Schuster.

Moursund, D., & Yoder, S. (1988, March, April, May, September). *Logo Exchange*. (A series of four articles on Logo and problem solving).

Nelson, T. (1987). *Computer lib/dream machines*. Redmond, WA: Tempus Books. (Originally published in 1974.)

Niedelman, M. S. (1990). *An investigation of transfer to mathematics of a problem-solving strategy learned in earth science*. Unpublished doctoral dissertation, University of Oregon, Eugene.

Polya, G. (1957). *How to solve it: A new aspect of mathematical method*. Princeton University Press.

Rubinstein, M. F. (1986). *Tools for thinking and problem solving.* New York: Prentice-Hall.

Salomon, G., & Perkins, D. (1987). Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research,* 3(2), 149-169.

Salomon, G., & Perkins, D. (1989). Rocky roads to transfer: Rethinking mechanisms of a neglected phenomenon. *Educational Psychologist, 24*(2), 113-142.

Steinberg, E. R., Baskin, A. B., & Hofer, E. (1986). Organizational/memory tools: A technique for improving problem solving skills. *Journal of Educational Computing Research,* 2(2), 169-87.

Sternberg, R. J. (1988). *The triarchic mind: A new theory of human intelligence.* New York: Penguin Books.

Sternberg, R. J. (1990). Thinking styles: Keys to understanding student performance. *Phi Delta Kappan, 71*(5), 366-371.

Yates, B. C., & Moursund, D. G. (1988). The computer and problem solving: How theory can support classroom practice. *The Computing Teacher, 16*(4), 12-16.

Yoder, S., & Moursund, D. (1990). *Introduction to Logo PLUS and problem solving.* Eugene, OR: International Society for Technology in Education.

Yoder, S., & Moursund. D. (1990). *Introduction to LogoWriter and problem solving.* Eugene, OR: International Society for Technology in Education.

Yoder, S. (1992). *HyperTalk for educators: An introduction.* Eugene, OR: International Society for Technology in Education.

Index

Accidental
  in music 119
Animation 61, 97, 127
  flipping cards 130
Answer
  format of 132
Background 31, 89
  description 31
  editing 89, 92
  getting blank 30
  in stack design 106
  keeping default 30
Beep sound 115
Browse 44
Browse tool
  defined 34
Brush 44
Bucket 44
Bugs 75, 123
  in computer programs 40
Button
  as components of stacks 63
  calculator 26
  conventions 68, 108
  copying 67
  deleting 67
  elevator 26
  icon 67
  linking 63
  new 63
  opaque 65, 94
  pasting 67
  script 69
  transparent 78, 83, 95
  with music 122
Button tool 44
Card
  new 32
Chess 18, 141
Chunking 39, 148
Click and drag 33
Clip art 22, 56
Close box 33
Communication
  in *HyperCard* 29
  nonsequential 74
Computer 127
  defined 1
  in problem solving 146, 147
Computer model 129
Computer program
  defined 1
Consistency 111
Constructivism 42
Constructs
  branching 133
  programming language 133
  repetition 133
  sequencing 133
Conventions
  in stack design 108
Copying
  button 67
  field 80
Curve 44
Debugging 40
Digitized sound 118
Display stacks 104
Domain specificity 60
  defined 18
Dotted notes 120
Draw Filled 50
Duration
  symbols to use with Play 120
Eraser 44
Errors
  logic 124
  syntax 123
  the computer can detect 123
  the computer cannot detect 124
Expert 146
  world class 141
FatBits 47
Feedback
  from computer 41
  from others 41
  from self 41

Field
as components of stacks 79
creating 79
text in 80
Field tool 44
Filter stacks 104
Flipping cards
to animate 130
Graphical user interface 45, 61
Grid 61
HyperCard
definition of 1
hardware needed iii
object-like programming 7
starting 21
uses in education 8
version of iii
Hypermedia 1
defined 5
reading and writing 5
Hyperspace 24
lost in 37
navigating in 24, 26
HyperTalk 1, 5
Hypertext 1
Icon 22
conventions for use 108
in stack design 108
If 133
Interactive programming 131
It 133
Journaling 19, 142
Jump linear stacks 100
Keep 58
Lasso 44
Layering of objects 81, 94
Learning
and doing 41
trial and error 24, 35, 40
Learning theory 42
Line 44
Linear stacks 100
Lost in Hyperspace
recovery from 111
Mathematical model 129
Mental model 24, 128
Message box
introduction to 116
moving 117
typing in 117
Metacognition 10, 19, 142, 144
Mind tools 7
Mind's eye 60, 75
Mistakes
as part of learning 40
Model 128
computer 129
mathematical 129
scale 129
written 128
Modeling 60, 128
Mouse 22
Music
chart of notes 120
stopping 122
Navigation 36
Network stacks 103
Notes
chart of 120
dotted 120
how represented 119
triplets 120
Object 74
in *HyperCard* 7
Object oriented programming 7
Octave
setting 119
Oval 44
Paint text 34
Pallet 49
Pencil 44
Pixel 61
Play 118
problems with 123
syntax 119
Point and click stack 104
Pointer 22
arrow 116
bucket 51
cross hairs 45, 49
eraser 48
I-beam 53, 116

lasso 46
paint brush 48
pencil 46
pointing hand 116
polygon 54
spray can 49
vertical bar 116
Polygon 44
Pop-up field 83
Precise thinking 145
Primitive problem 38
Problem
defined 15
primitive 38
Problem posing 10, 85
Problem representation 142
Problem situation 13, 142, 144
Problem solving
a thought experiment 24
aids to 7
build on previous work 25
by pushing buttons 26
cooperative 143, 147
domain dependence 146
five-step plan 17
in programming 148
persistence 85
positive attitude 148
prodigious memory 147
strategy 19
talking about 142
Programming language 38, 127
constructs 133
higher level 127
Recent
representation of cards 37
Rectangle 44
Regular polygon 44
Reinventing the wheel 25
Repeat 131
Rest
in music 119
Revert 58
Round rectangle 44
Saving a stack 38
Scale model 129
Scripting a button 69
Select 44
Short term memory 148
Single frame stacks 104
Sound
available in *HyperCard* 118
beep 118
boing 118
flute 118
harpsichord 118
setting 118
Spray can 44
Stack i, 2, 31
creating a new 29
design conventions 108
map 3
new 31
point and click 104
structure of 99
Stacks
designing 99
display 104
filter 104
jump linear 100
linear 100
network 103
single frame 104
tree structure 101
Stop
with Play 122
Stopping music 122
Strategy 19, 144
Template 77, 97
Tempo
with Play 121
Text 44
in fields 80
paint 34, 80
paint versus text in fields 80
Text tool
using 34
Tool names 44
Tool window
tearing off 34
using 33

Tools
    using 35
Trackball 22
Transfer of learning 18
    High Road 18, 24
    Low Road 18, 24
Tree structure
    of stacks 101
Triplets 120
User level
    setting 29
Variable 133
Version of *HyperCard* iii
Visual
    command 139
    effect names 139
    image 140
    speed 139
    syntax 139
Visual effect 72, 139
    barn door 106
    checkerboard 106
    dissolve 105
    in stack design 105
    iris 106
    scroll 105
    shrink 106
    speed 73
    stretch 106
    wipe 106
    zoom 106
World class expert 141
Written model 128

Put the Future of Education in Your Hands...

Edutrends 2010
Restructuring, Technology, and the Future of Education
Education, Technology, and Paradigms of Change for the 21st Century
David D. Thornburg, Ph.D.

Pick up David Thornburg's visionary books that explore the social and technological trends shaping education.

Edutrends 2010—Restructuring, Technology, and the Future of Education

Thornburg examines the recent flurry of teacher bashing, moving from the reasoning flaws of those who would dismantle U.S. public education to the change process necessary for educational restructuring for the 21st century.

He proposes a way to think intelligently about the future while we reshape education to meet the needs of our society.

Education, Technology, and Paradigms of Change for the 21st Century

Learn unique patterns of thinking that encourage students to go above and beyond the three Rs. The technologies of calculators, computers, laser discs, and video games converge with new concepts of learning and intelligence to provide unprecedented opportunities for all students. Let this progressive book lead you toward a future filled with new paradigms.

International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

All you need is the right tool...

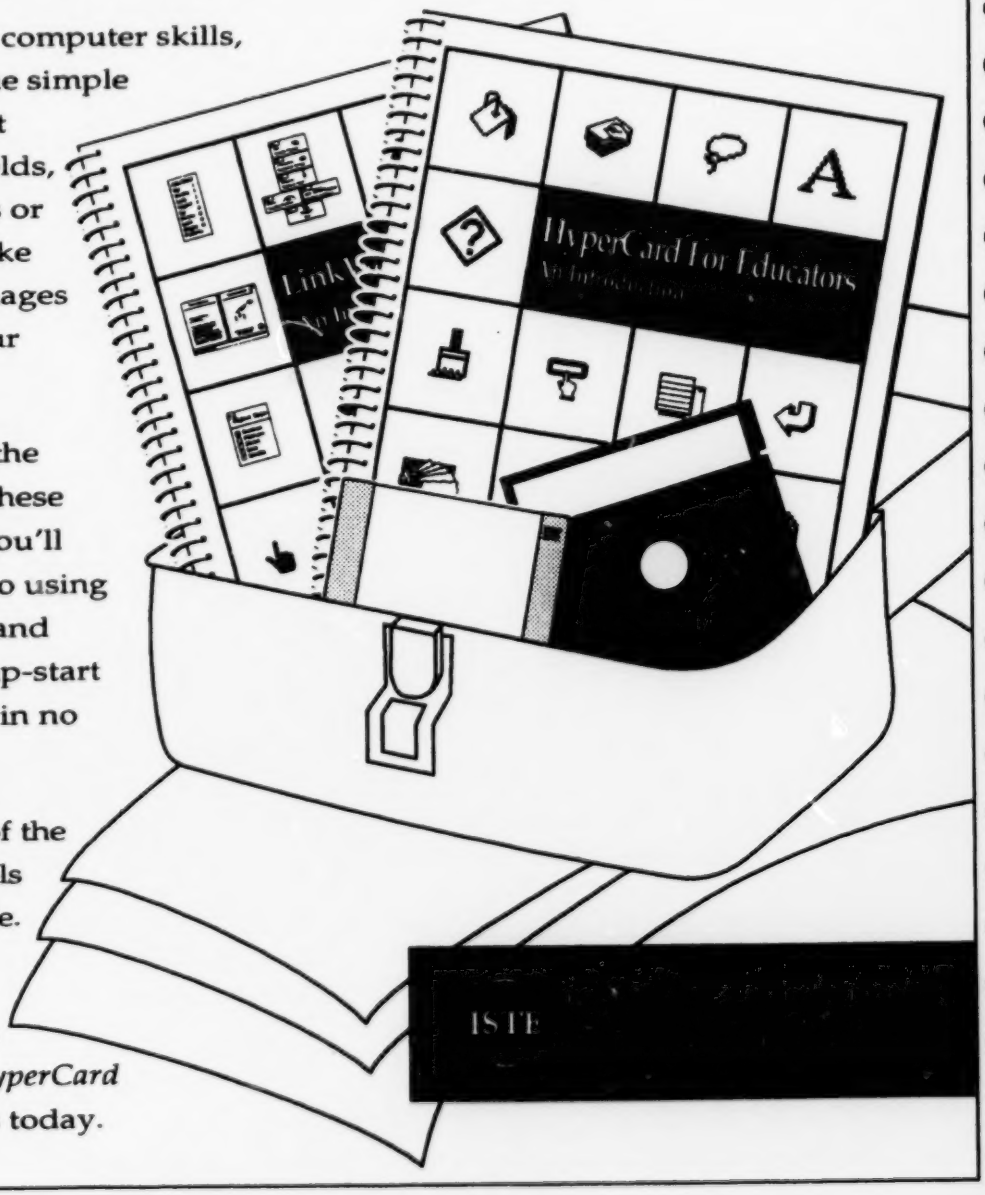
With either *HyperCard for Educators* (Macintosh) or *LinkWay for Educators* (MS-DOS) as an introduction to the interactive world of hypermedia, you can learn to build libraries of knowledge suitable for both classroom and personal use.

If you have basic computer skills, you can follow the simple tutorials and start using buttons, fields, and backgrounds or base pages to make your data and images come alive in your classroom.

By starting with the foundation that these books provide, you'll be on your way to using sounds, images, and animation to jump-start your curriculum in no time!

Take advantage of the organizational tools computers provide.

Call ISTE to order and begin building your *HyperCard* or *LinkWay* skills today.



Make This Step Your Next Step.

You've got your new Mac... Now what?

You want to know what those cute icons mean. You need to find out how to use that mouse-thing-a-ma-jig.

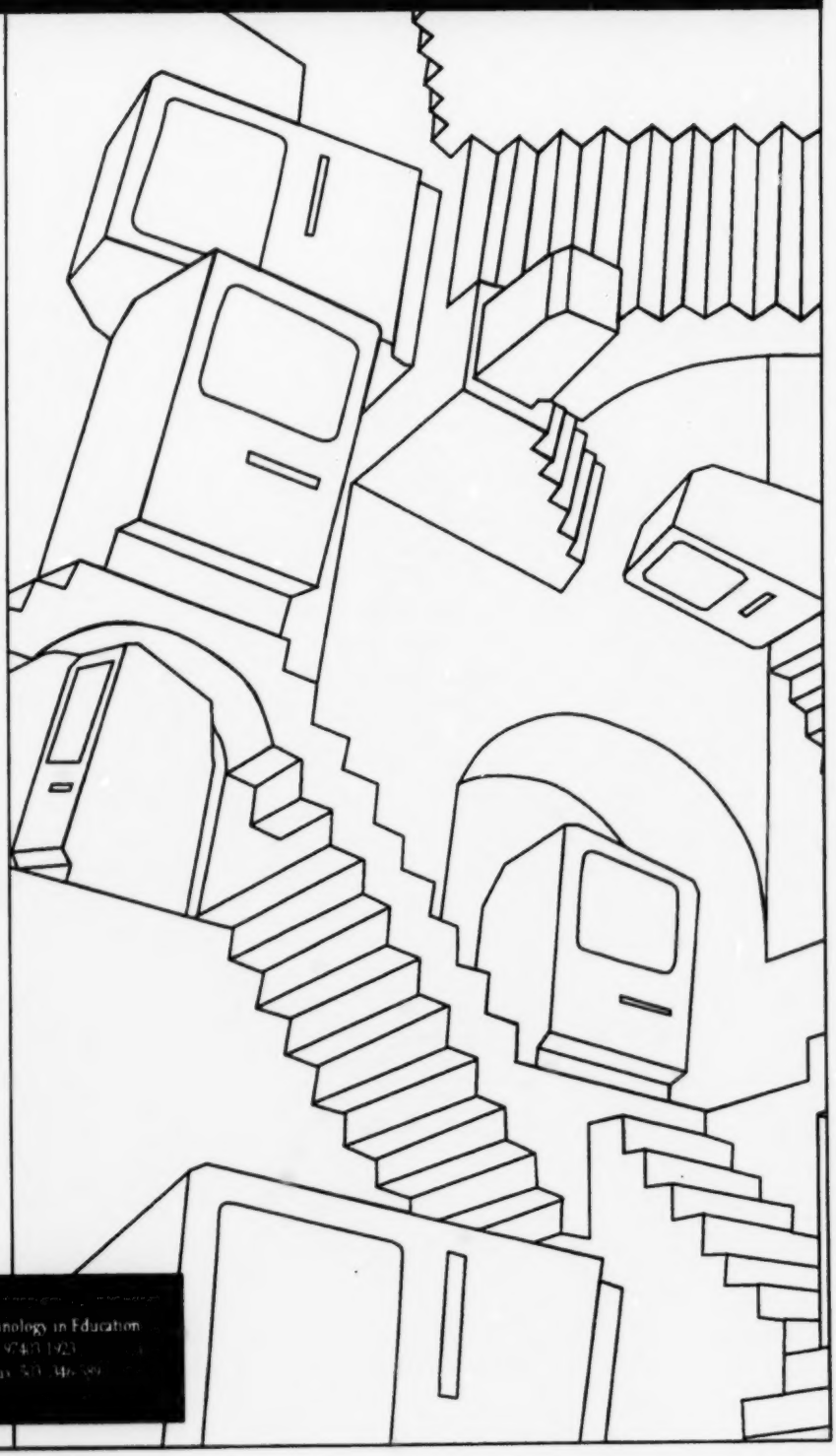
You'd like to know how to "empty the trash," and it'd be great to understand exactly what the "chooser" chooses.

By following the footsteps in ISTE's new book, *Macintosh Step by Step*, you'll pick up on all these basic Macintosh operating procedures, plus more—from finding the "power" button to demystifying the little Apple menu in the upper left corner.

And it also teaches you the theory behind Mac file storage and the importance of backing up your data on a regular basis.

In all, you'll find that this simple, illustrated guide will lead you on your way toward becoming a Mac user—not just a Mac owner.

Just call today and make your next computing step a step in the ISTE direction.



ISTE International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800-336-5191 Fax: 503-346-5890

ISTE Books & Courseware Order Form

Name _____

School/Business _____

Address _____

City _____ State _____ Zip/Postal Code _____

Country _____ Phone _____

☐ Yes! I'm presently an ISTE member. Membership number must be included to receive discount. My membership number is _____ (Discount does not apply to initial membership fee.)

Occupation

Grade Level: ☐ all (K-12) Specific grade level _____

- ☐ Teacher (350) Subject _____
- ☐ Administrator (320) Title/Position _____
- ☐ Higher Education (360) Title/Position _____
- ☐ Federal/State Government (310) Title/Position _____
- ☐ Educational Information Resource Manager (IRM) (320)
- ☐ Technology Coordinator (340)
- ☐ Other _____

Terms & Conditions

Terms: Net 60 orders must be accompanied by authorized company purchase orders. Orders for individuals must be pre-paid, C.O.D., or charged to a Visa, MasterCard, or Discover card number. All non-U.S. orders must be pre-paid.
Shipping: Shipping method is UPS. Add 5% in addition to shipping rate for first class delivery. ***Allow three weeks for delivery.*** Call for rush orders.
Prices: Subject to change without notice.
Returns: ISTE allows return of merchandise, provided the materials are in resellable condition. There will be a 10% restocking charge. All returns must be accompanied by an ISTE Return Authorization form. If you wish to return merchandise, please contact the Distribution Dept. *GST Registration #128828431*

Shipping & Handling for Books & Courseware

$0-$15.99 (subtotal) ..... add $3.25
$16-$45.99 (subtotal) $4.50
$46-$75.99 (subtotal) $5.50
$76-$100.99 (subtotal) $6.50
$101 or more ..... 7% of subtotal

Do not include *additional site license fees or subscription costs* when computing shipping rates.

Non-U.S. orders for Books & Courseware are sent surface mail. If you want your Books & Courseware order shipped AIRMAIL, please check here. ☐ ISTE will bill you the additional shipping charge.

Payment Options

☐ **Payment enclosed.** (Make checks out to ISTE.) Non-U.S. orders must be pre-paid with U.S. funds or credit card.
☐ VISA ☐ MasterCard ☐ Discover Card Exp. Date _____

☐ **Purchase Order enclosed.** (Please add $2.50 for order processing. P.O. not including $2.50 fee will be returned.)

☐ **C.O.D.** for U.S. Book orders only. You will pay UPS the total upon delivery (check or cash—ISTE will add $2.75 order processing).

GST Registration Number 128828431

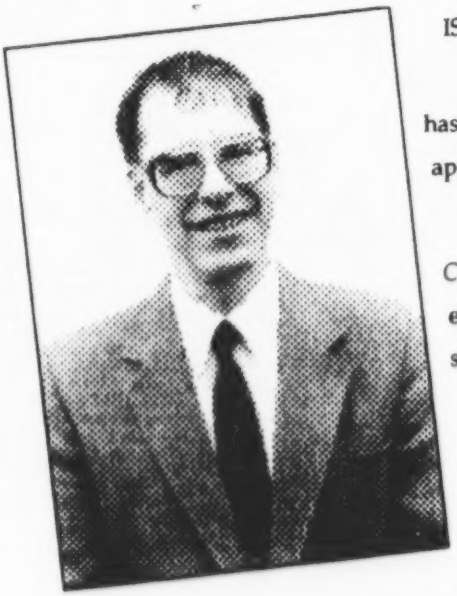
ISTE Books, Courseware, & Nonmember Subscriptions

Qnty.	Title	Member Unit Price	Nonmember Unit Price	Total Price

BOOKS & COURSEWARE SUBTOTAL	
Add Shipping, based on Books & Courseware SUBTOTAL	+
Add $10 S & H for Don't Copy that Floppy	+
Add Additional 5% of SUBTOTAL if shipped to PO Box, AK, HI, or outside U.S.	+
Add 7% of SUBTOTAL for GST if shipped to Canada	+
If billed with purchase order, add $2.50; If COD, add $2.75	+
	=
Membership Total from other side	+
TOTAL	=

MAIL this form to:
ISTE, 1787 Agate St., Eugene, OR 97403-1923 USA
FAX this form with credit card or P.O. information to: 503/346-5890
PHONE your credit card or P.O. order to:
ISTE Order Desk: 800/336-5191

Dave Moursund knows...



ISTE Executive Officer Dave Moursund is dedicated to improving education.

Since he first began working with computers in education in 1963, Dave has constantly sought to improve education at all levels through the use of appropriate computer-related technology.

As a continuing part of this effort, Dave's newest book—*The Technology Coordinator*—discusses the goals of education and how teachers and teachers of teachers can join in the enormous change process facing today's schools.

This book was written for educators who want to play a leadership role in the instructional use of technology. Dave uses interviews with technology coordinators to illustrate the needs, roles, and qualifications needed to accomplish the goals of educational technology.

In recent years, Dave has focused much of his writing efforts on problem solving and on staff development. Some of Dave's other books also available from ISTE include *Problem-Solving Models for Computer Literacy, LogoWriter for Educators: A Problem Solving Approach, Logo PLUS for Educators: A Problem Solving Approach* (co-authored with Sharon Yoder), and *Effective Inservice for Integrating Computer-as-Tool into the Curriculum.*

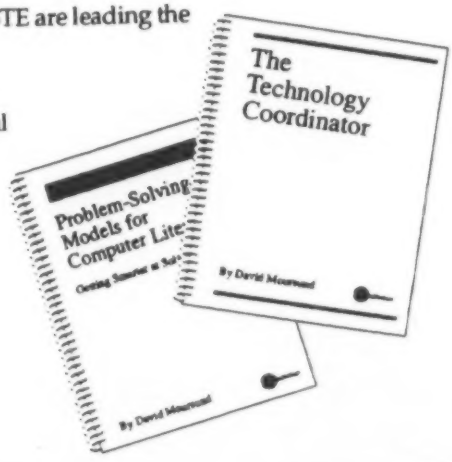
"Our educational system is not nearly as good as it could be," says Dave. "We know that computer-related technology can make substantial contributions to improving both the content of the curriculum and the instructional delivery system. Such technology can empower students and teachers."

ISTE believes that computer-related technology must become a tool that students and teachers use routinely if students are to be adequately prepared for adult citizenship in our Information Age society.

ISTE's wide range of periodicals, publications, projects, and other activities mirrors this vision and commitment.

Call us today to order Dave's latest books, and discover how he and ISTE are leading the way for the educators and school systems of tomorrow.

ISTE. Proudly putting you in touch with the world of educational technology.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

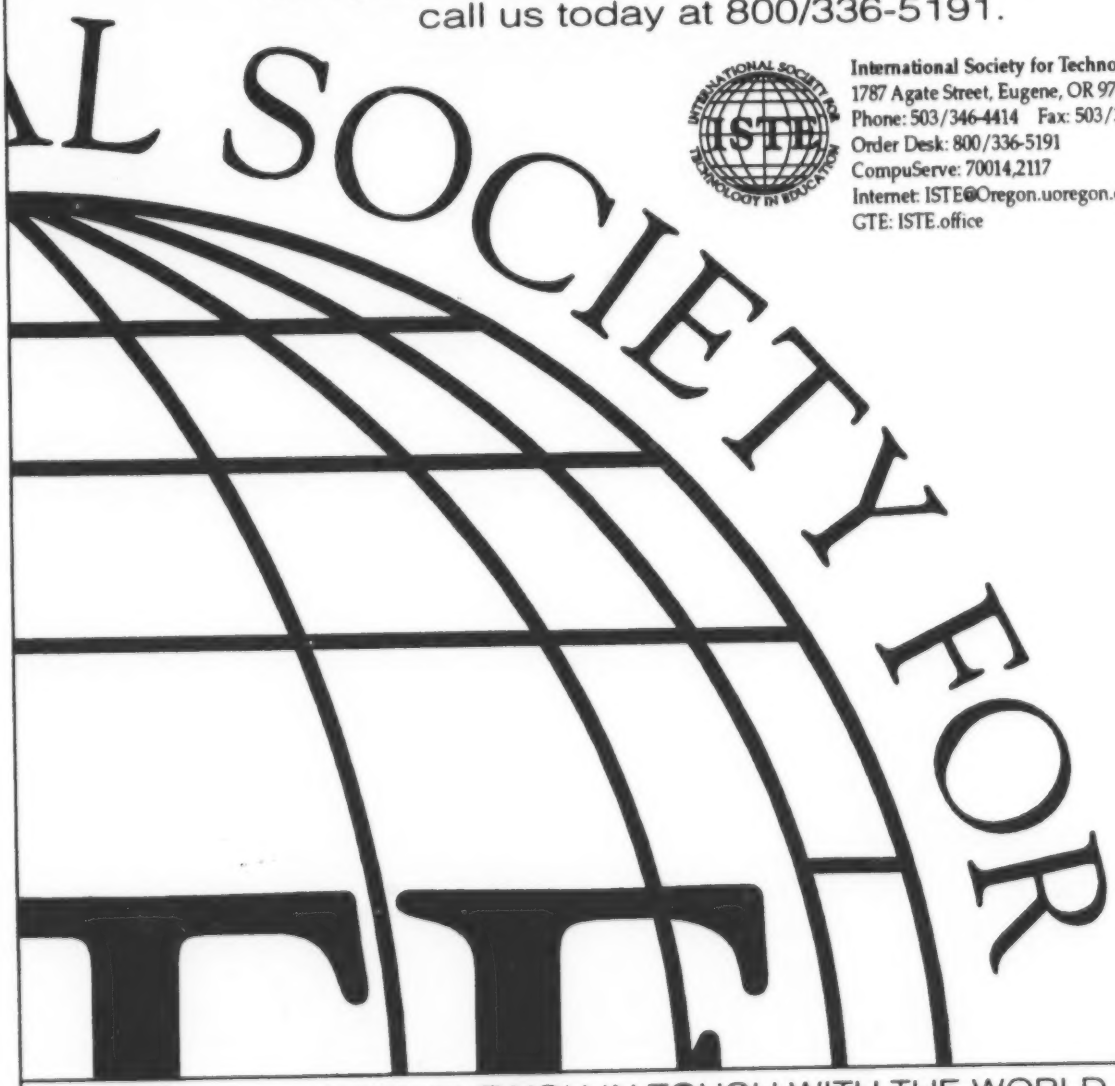
ISTE BRINGS THE WORLD OF TECHNOLOGY CLOSER TO YOU.

By drawing from the resources of committed professionals worldwide, ISTE provides support that helps educators like yourself prepare for the future of education.

ISTE members benefit from the wide variety of publications, specialized courseware, and professional organizations available to them.

They also can enjoy exciting conferences, global peer networking, and mind-expanding independent study courses.

So if you're interested in the education of tomorrow, call us today at 800/336-5191.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Phone: 503/346-4414 Fax: 503/346-5890
Order Desk: 800/336-5191
CompuServe: 70014,2117
Internet: ISTE@Oregon.uoregon.edu
GTE: ISTE.office

WE'LL PUT YOU IN TOUCH WITH THE WORLD.